

Grado en Ingeniería de Sistemas Audiovisuales
2016-2017

Trabajo Fin de Grado

Comunicaciones Sincronizadas en Dispositivos Limitados

Carlos Parra Marcelo

Tutor

Ignacio Soto Campos

Leganés, 10 de octubre de 2017



Esta obra se encuentra sujeta a la licencia Creative Commons
Reconocimiento – No Comercial – Sin Obra Derivada

Trabajo de Fin de Grado:	Comunicaciones Sincronizadas en Dispositivos Limitados
Autor:	Carlos Parra Marcelo
Tutor:	Ignacio Soto Campos
Fecha:	10 de octubre de 2017

Tribunal

Presidente:	Pablo Basanta Val
Secretario:	Álvaro Montero Montes
Vocal:	Iván González Díaz
Suplente:	Carlos García Rubio

Agradecimientos

A mis padres, por haberme educado y haberme dado la posibilidad de llegar hasta aquí. Aunque no os lo agradezca muy a menudo, gracias.

A mi pareja, por ser el apoyo constante que cualquiera necesita para convertir los retos en éxitos. Seguro que vendrán más.

A mi tutor Ignacio Soto, por mostrarse tan atento con mis dudas desde el primer momento y por su disponibilidad para ayudarme o aconsejarme.

A mi familia, a mis amigos y en general a toda la gente que me ha apoyado durante este trabajo y toda la carrera.

Resumen

En este trabajo se diseña y prueba un sistema de comunicación síncrono para los dispositivos limitados autónomos que componen las redes de sensores inalámbricas. El objetivo que se persigue con ello es lograr un notable ahorro de energía en las baterías de los nodos sin que éstos pierdan la capacidad de envío y recepción de información. Para ello se programan períodos de intercambio de datos y desactivando el módulo de comunicaciones fuera de estas etapas de tiempo.

Para la creación de la red de sensores se ha elegido la plataforma modular *opensource* Wasp mote y el estándar de comunicación IEEE 802.11, popularmente conocido como Wi-Fi, debido a su gran flexibilidad.

Previo al diseño del sistema se ha realizado un breve estudio de la situación actual de las redes de sensores inalámbricas: esquemas más utilizados, tecnologías de comunicación más apropiadas, proyectos actuales, etc.

Con ayuda de los datos recopilados en dicha investigación se ha propuesto un diseño teórico organizado mediante una topología tipo árbol en la que un extremo del sistema lo ocupa un servidor con acceso a una base de datos y el otro extremo está compuesto por un conjunto de nodos sensores inalámbricos. Entre ellos, un *sink node* hace de enlace entre el resto de Wasp motes de la red y el servidor. Organizado de esta manera, el sistema es capaz de configurarse completamente al descargar los ajustes necesarios del servidor, y los sensores quedan programados para establecer conexiones periódicas nodo-nodo y *sink node* - servidor con el fin de almacenar las mediciones realizadas en la base de datos. De esta forma, los Wasp motes consiguen ahorrar energía utilizando el modo *sleep* en los períodos de inactividad.

Posteriormente, se ha desarrollado un prototipo basado en el diseño propuesto con el fin de comprobar el ahorro de energía conseguido respecto a otros diseños. Los resultados obtenidos demuestran que la solución sincronizada propuesta consigue aumentar la autonomía de las baterías de los nodos sin perjudicar el régimen de comunicación habitual de las redes de sensores.

Abstract

In this work is designed and tested a synchronous communication system for the autonomous limited devices that currently compose the wireless sensor networks. The objective is to achieve a remarkable energy saving in the nodes' batteries by programming periods of data exchange and disabling the communications module out of these stages of time, but without losing the ability to send and receive information.

For the creation of the sensor network has been chosen the modular opensource platform Waspote and the IEEE 802.11 communication standard, popularly known as Wi-Fi, due to its great flexibility.

Before the system's design, a brief study of the wireless sensor networks' current situation was made: the more used schemes, the more appropriate communication technologies, current projects, etc.

Using the information obtained, a theoretical design, organized in a tree topology, was proposed, in which one end of the system is occupied by a server with access to a database and the other end is composed of a set of wireless sensor nodes. Among them, a sink node makes a link between the Waspotes and the server. Organized this way, the system is able to configure itself by downloading the necessary settings from the server, and the sensors are programmed to establish periodic node-node and sink node - server connections in order to store the made measurements in the database. In this way, the Waspotes manage to save energy using the sleep mode during downtime periods.

Later, a specific test environment has been developed in order to check the energy saving achieved respect to other designs. The results show that the proposed synchronized solution manages to increase the batteries' autonomy without damaging the usual communication regime of sensor networks.

ÍNDICE

1. INTRODUCCIÓN	8
1.1. Introducción.....	8
1.2. Propósito y objetivos	9
1.3. Estructura de la Memoria.....	10
2. ESTADO DEL ARTE	11
2.1. Red de sensores inalámbricos	11
2.1.1. Origen.....	11
2.1.2. Características generales.....	12
2.2. Waspmotes.....	14
2.3. Técnicas para la optimización de energía	16
2.3.1. Gestión y división del tiempo.....	17
2.3.2. Técnicas orientadas a datos.....	17
2.3.3. Técnicas basadas en la movilidad de los sensores	18
2.4. Sincronización de hora entre nodos sensores inalámbricos	18
2.5. Aplicaciones de las WSN	20
2.5.1. Monitorización ambiental	20
2.5.2. Cuidados médicos.....	20
2.5.3. Posicionamiento y seguimiento de objetos o seres vivos	21
2.5.4. Logística	21
2.5.5. Entorno doméstico	22
2.5.6. Entorno industrial y comercial	22
2.5.7. Otros campos de interés	23
2.6. Trabajos relacionados.....	24
2.7. Grandes proyectos	25
2.8. Marco regulador	27
2.8.1. Estándares técnicos	27
2.8.2. Análisis de la legislación aplicable a la solución implementada	30
3. DISEÑO DE LA SOLUCIÓN PROPUESTA.....	32
3.1. Descripción del diseño propuesto.....	32
3.2. Requisitos generales.....	33
3.3. Descripción del sistema implementado.....	33
3.3.1. Hardware y software utilizado	33

3.3.2. Esquema de conexiones	34
3.4. Desarrollo del sistema implementado.....	36
3.4.1. Red de sensores inalámbricos (WSN)	36
3.4.2. Servidor HTTP	37
3.4.3. Mecanismo de configuración de la WSN.....	40
3.5. Análisis del código implementado en los Waspmotes	41
3.5.1. Ajustes iniciales	41
3.5.2. Bibliotecas importadas y constantes y variables globales.....	42
3.5.3. Descripción breve de las funciones implementadas	43
3.6. Resumen del capítulo	46
4. PRUEBAS DE FUNCIONAMIENTO.....	47
4.1. Pruebas realizadas.....	47
4.1.1. Ajustes previos	47
4.1.2. Registro de pruebas	48
4.2. Resultados obtenidos	49
4.2.1. Prueba de comprobación del ahorro de energía	50
4.3. Conclusiones del capítulo.....	53
5. CONCLUSIONES.....	54
5.1. Conclusión principal	54
5.2. Conclusiones secundarias	54
5.3. Posibles mejoras	56
5.3.1. Proposición de un sistema más avanzado.....	58
6. BIBLIOGRAFÍA	59
ANEXO I: Resumen extendido en Inglés.....	63
ANEXO II: Entorno socio-económico	69
Presupuesto del Trabajo de Fin de Grado	70
Impacto socio-económico.....	72
ANEXO III: Código implementado.....	74
Diagrama de flujo del programa principal	74
Código C++ del programa principal.....	78
Código PHP 5.6 del archivo “config.php”	92
Código PHP 5.6 del archivo “dataServer.php”	93
ANEXO IV: Registro de mediciones.....	94

1. INTRODUCCIÓN

1.1. Introducción

La aparición de las WLAN (*Wireless Local Area Network*) en el sector de las telecomunicaciones ha marcado un antes y un después en lo que se refiere al acceso a internet.

Anteriormente, el usuario se solía conectar a la red mediante un ordenador de sobremesa, ya que era necesaria una conexión por cable. Los ordenadores portátiles ya se utilizaban, pero no eran tan comunes en el entorno doméstico ya que no se podía explotar su movilidad a la hora de acceder a la red.

Actualmente, la variedad de dispositivos que pueden conectarse a una WLAN es inmensa y su desarrollo se ha visto potenciado gracias a las posibilidades que aporta el acceso inalámbrico. No solo es más común que el usuario se decida por un ordenador portátil antes que por uno de sobremesa, sino que ahora puede elegir qué dispositivo le conviene comprar en función del uso que vaya a hacer de la red.

Además de la comercialización de nuevos dispositivos, tales como *smartphones*, tabletas o *netbooks*, también se han desarrollado otros que no están enfocados a la interacción con el usuario. En este último grupo se incluyen algunos pequeños dispositivos electrónicos autónomos, cuya misión es únicamente recopilar información o monitorizar el curso de uno o varios parámetros fisiológicos, ambientales, o de otra naturaleza. A la interconexión digital de este tipo de dispositivos y otros objetos cotidianos se le ha denominado como IoT (*Internet of Things*).

Este trabajo está relacionado con algunas de estas pequeñas unidades electrónicas que, en conjunto, son capaces de formar redes de sensores inalámbricas o WSN (*Wireless Sensor Networks*). Estas redes se componen de dispositivos distribuidos autónomos y cada uno de ellos incorpora al menos un sensor con el que puede monitorear condiciones físicas o medioambientales. No obstante, la interconexión inalámbrica de este tipo de dispositivos requiere una cantidad de energía que puede suponer un problema según el tiempo que se necesite para realizar las sucesivas mediciones, ya que su autonomía es limitada si funcionan mediante baterías (algo habitual). Por esta razón es necesario encontrar un equilibrio entre prestaciones y ahorro de energía, para así obtener un sistema de monitorización eficiente y que sea capaz de estar operativo durante un período de tiempo que se ajuste a las necesidades de su aplicación.

Una forma fácil de ahorrar batería es apagar el módulo de comunicaciones del sensor mientras no se envían ni se reciben datos. Sin embargo, en muchas aplicaciones se necesita que un dispositivo pueda recibir comunicaciones de otras unidades pertenecientes al sistema. Si en ese instante concreto el módulo de comunicaciones del dispositivo está apagado, en ningún caso se podrá establecer una comunicación.

En este trabajo se ha desarrollado una solución al problema que se acaba de mencionar basada en sincronizar los dispositivos inalámbricos para que, en un momento determinado, se conecten entre ellos y puedan transferirse información. De esta forma se consigue ahorrar energía durante los períodos de tiempo en los que no se producen comunicaciones entre los nodos que conforman el sistema.

1.2. Propósito y objetivos

La finalidad de este Trabajo de Fin de Grado es desarrollar y probar una solución, lo más sencilla posible, para que los dispositivos sensores limitados de una WSN sean capaces de transferir las mediciones tomadas a una base de datos de forma sincronizada y minimizando el coste energético durante todo el proceso. Para su implementación se utiliza una plataforma modular *opensource* llamada Wasmote, que permite construir redes de sensores inalámbricas de muy bajo consumo [1].

El sistema de sincronización de comunicaciones debe ser capaz de ahorrar energía mediante el uso del modo *sleep* disponible en los Wasmotes durante los períodos significativos de tiempo en los que no se produzcan comunicaciones, ya que al estar todos los módulos apagados (entre ellos el de comunicación) el consumo energético se reduce considerablemente. A su vez, la solución sincronizada debe de ser capaz de garantizar una forma de comunicación periódica entre los nodos durante el tiempo de operatividad que permitan las baterías de los Wasmotes.

Además, la solución propuesta debe cumplir algunos objetivos más concretos:

- El sistema debe soportar comunicaciones tipo *ad-hoc* pero también debe implementar la configuración necesaria para que los nodos utilicen una red de infraestructura. Asimismo, las comunicaciones en la red deben *peer-to-peer*, ya que ambos participantes de una conexión pueden pedir servicios y ofrecerlos.
- La red debe disponer de acceso a una base de datos para dar la posibilidad de almacenar la información recopilada a través de los sensores. Como consecuencia, el sistema debe de estar preparado para hacer llegar las mediciones de los sensores desde los Wasmotes hasta dicha base de datos haciendo uso de comunicaciones periódicas sincronizadas.
- Los sensores deben ser configurables (al menos parcialmente) desde un servidor web sin necesidad de modificar el código implementado en los mismos.
- El sistema propuesto debe ser probado posteriormente a su diseño y elaboración para demostrar la validez de la solución síncrona implementada y su ventaja en la autonomía de la red respecto a otros sistemas asíncronos que no apagan sus módulos durante los tiempos de inactividad.

1.3. Estructura de la Memoria

Este documento está dividido en seis capítulos y cuatro anexos en los que la información queda organizada de la siguiente forma:

- **Capítulo 2. Estado del arte:**

Se realiza un análisis de algunos conocimientos teóricos básicos para una mejor comprensión del contexto del trabajo. También se analiza la situación actual del entorno en el que se desarrolla el proyecto y se mencionan otros trabajos relacionados.

Al final de este capítulo se incluye el apartado **Marco regulador**.

- **Capítulo 3. Diseño de la solución propuesta:**

Se describe el sistema propuesto y se detallan los requisitos generales exigidos. A continuación, se describe la implementación de un prototipo de red de sensores inalámbricos que sigue la solución propuesta y se explican las fases que han sido necesarias para su diseño y desarrollo. También se incluye toda la información del funcionamiento de los archivos de código implementados. Finalmente se incluye un pequeño resumen a modo de conclusión.

- **Capítulo 4. Pruebas de funcionamiento:**

En este capítulo se explican las pruebas efectuadas con el fin de comprobar el correcto funcionamiento del sistema diseñado y se detallan los resultados obtenidos. Al final se incluye un pequeño resumen a modo de conclusión.

- **Capítulo 5. Conclusiones:**

Se analizan en profundidad las conclusiones extraídas del entorno de pruebas y, por último, se proponen posibles mejoras a implementar en el sistema como futuras vías de desarrollo.

- **Capítulo 6. Bibliografía.**

- **Anexo I: Resumen extendido en Inglés.**

- **Anexo II: Entorno socio-económico.**

- **Anexo III: Código implementado.**

- **Anexo IV: Registro de mediciones.**

2. ESTADO DEL ARTE

2.1. Red de sensores inalámbricos

Una red de sensores inalámbricos o WSN (*Wireless Sensor Network*) es una red inalámbrica a la que están conectados un conjunto de dispositivos distribuidos autónomos que utilizan sensores para monitorear condiciones físicas o ambientales. Además, dicho sistema suele incorporar un *gateway*, que es un nodo que actúa como enlace entre los dispositivos autónomos y el resto de la red cableada (conecta dos redes de distinto tipo).

Estas redes se suelen organizar típicamente en 3 topologías: estrella, árbol o malla. En el primer tipo de topología mencionada los dispositivos autónomos solo pueden establecer conexión con el *sink node*, como se puede apreciar en la siguiente imagen [2]:

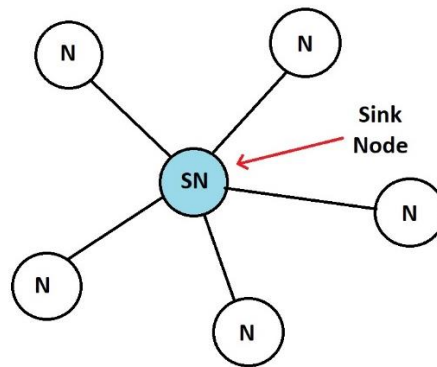


Ilustración 1. Topología en estrella.

Las WSN son redes de procesamiento distribuido, ya que los dispositivos realizan las mediciones pero no las procesan (o al menos no realizan todo el procesamiento). Al llegar al *sink node* los datos se pueden procesar o se pueden almacenar en una base de datos, que puede estar en otra red diferente pero siempre accesible a través del *gateway*.

2.1.1. Origen

La investigación de las redes de sensores inalámbricos (WSN) comenzó a principios de este siglo, gracias a los avances científicos en relación a las tecnologías de transmisión inalámbrica que se fueron produciendo en la segunda mitad del siglo XX [3]. La principal ventaja que tienen estas redes inalámbricas frente a las cableadas es la flexibilidad que adquieren ya que los nodos no tienen que estar permanentemente conectados a la corriente y, en su lugar, utilizan baterías como fuente de energía. Por este motivo, las WSN fueron captando el interés de la industria, lo que propició un incremento de proyectos de investigación y, como consecuencia directa, un gran desarrollo. La posibilidad de apertura a nuevos mercados, debido a las potenciales aplicaciones de este tipo de sistemas, también influyeron en este aspecto. Sin embargo, el reto que siguen planteando estas redes en la actualidad es la optimización de la energía, para así poder aumentar la autonomía de los dispositivos que las conforman.

En la aparición de las WSN tuvieron gran influencia las redes *ad-hoc*, ya que los nodos de las redes de sensores se conectan principalmente mediante topologías de este tipo. En la siguiente tabla se puede observar una breve comparativa entre una red de sensores y una configuración tipo *ad-hoc* estándar [4]:

	WSN	Redes ad hoc
Densidad de red	Alta	Baja/Media
Probabilidad de interferencia	Alta	Baja/Media
División de la red	Posible	Improbable
Recursos	Limitados	Aceptables
Tipo de comunicación	Distribuida	Peer-to-peer
Fuente de energía	Irreemplazable	Remplazable

Tabla 1. WSN vs. *ad-hoc*. Fuente: [4]

2.1.2. Características generales

Las WSN pueden estar compuestas de dispositivos que varíen su posición con el paso del tiempo (ya que su diseño sencillo y ligero lo permite). De igual forma, no todos los nodos que formen parte del sistema tienen que tener el mismo hardware [3].

La posición geográfica del nodo, por muy extensa que sea el área que tengan que cubrir los dispositivos, también puede proporcionar beneficios. Uno de ellos sería la posibilidad de recargar la batería mediante energía solar o eólica si la localización lo permite. Sin embargo, en previsión de condiciones meteorológicas adversas, los nodos siempre deben llevar una protección capaz de aguantar, por ejemplo, tormentas de arena si el dispositivo se sitúa en el desierto [4].

En cuanto a la distribución de la red, la topología en estrella simplifica enormemente el procesamiento distribuido, pero puede tener ciertos problemas cuando la cantidad de nodos es grande y el *sink node* debe de soportar gran cantidad de conexiones, ya que puede gastar rápidamente su batería o sufrir un fallo y dejar el sistema completamente inoperativo. Por esta razón, existen redes de sensores en las que hay varios *sink nodes* y donde un protocolo correctamente diseñado indica a cada nodo a cuál le tiene que enviar sus paquetes de datos. Esta elección, que complica el funcionamiento del protocolo, se puede basar en diferentes parámetros: retraso mínimo, máximo rendimiento, mínimo número de saltos entre nodos [4], etc.

A continuación, se muestra una imagen con los dos esquemas de red que se acaban de explicar:

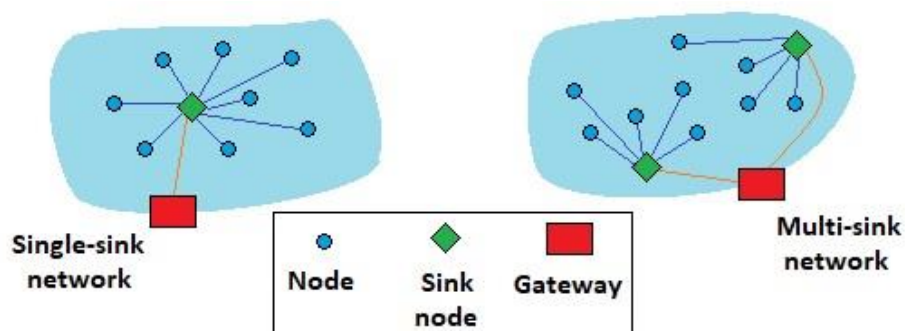


Ilustración 2. Esquemas de conexión.

Otro de los objetivos principales para con las WSN es su estandarización, que garantiza la interoperabilidad entre distintos equipos. En relación a las comunicaciones, el estándar IEEE 802.15.4 es el más extendido debido a su baja latencia y a su mayor flexibilidad en cuanto a requisitos mínimos [3], pero los estándares Wi-Fi o Bluetooth también son muy a menudo utilizados.

Las potenciales aplicaciones de las redes de sensores inalámbricas a la vida real son prácticamente ilimitadas, pero se pueden clasificar en dos tipos según el tipo de datos que obtienen [3]:

- Detección de eventos:

Este tipo de aplicación se basa simplemente en la comparación de las mediciones tomadas con un valor límite inferior y/o superior (*threshold*). Aunque el funcionamiento es muy simple (se podría decir que el resultado es binario), este sistema requiere un estudio probabilístico previo a la elección de dicho valor límite en el que se tengan en cuenta las consecuencias de una falsa alarma o de una pérdida.

- Estimación espacial y temporal de un proceso:

Como su propio nombre indica, estos sistemas recopilan información de un fenómeno físico concreto como, por ejemplo, la cantidad de CO₂ en el recinto de una fábrica o la temperatura de un reactor nuclear. En este caso cobra gran importancia la gestión de los datos que se van obteniendo a lo largo del tiempo, ya que tal cantidad de información debe ser interpretada correctamente realizando una selección lo suficientemente representativa.

Algunos sistemas reúnen características de los dos tipos de redes con el fin de aprovechar al máximo la instalación. Un ejemplo podría ser una red inalámbrica de sensores capaces de monitorizar la actividad sísmica en una zona y a su vez alertar de un posible terremoto.

Por un lado, los requisitos mínimos más comunes que se exigen a una WSN que detecta algún evento o fenómeno físico son los siguientes [3]:

- Eficiencia energética:

El ahorro de batería se puede conseguir en diferentes niveles del sistema, empezando por el hardware (componentes que tengan un bajo consumo), siguiendo por la capa física (módulo de comunicación), y terminando por el MAC (*Media Access Control*) o protocolos de enrutamiento.

- Baja velocidad de datos:

Esto es debido a que la información de los sensores es bastante sencilla y no se producen grandes flujos de datos.

- Comunicación en un sentido:

La información únicamente se dirige desde los sensores hasta la unidad principal de procesamiento o la base de datos, nunca en sentido contrario.

En el lado contrario, las redes de sensores utilizadas para monitorear un determinado proceso demandan las siguientes características [3]:

- Conexión robusta:

Garantías frente a interferencias (otras conexiones en la misma banda de frecuencias o condiciones ambientales adversas) y errores (fallos producidos en el hardware por diferentes motivos). Una red puede ser declarada inactiva debido a diferentes consideraciones, desde la pérdida de un nodo o el agotamiento de batería del 50% de la red, hasta la desaparición de un área de detección [4].

- Seguridad:

La red debe ofrecer ciertas prevenciones y soluciones para defenderse de ataques malintencionados.

- Interoperabilidad:

Que tanto los dispositivos autónomos como la unidad principal de procesamiento sean capaces de intercambiar información y utilizarla, es decir, que exista comunicación en ambos sentidos y que los nodos tengan más funciones y no únicamente la de recopilar datos.

- Alta velocidad de datos:

La información transmitida puede tener un carácter más complejo y, además, la cantidad suele ser mayor en este tipo de redes.

También, cabe la posibilidad de que los *sink nodes* puedan ir conectados al *gateway* de forma cableada o que la autonomía de los nodos siga teniendo gran importancia en el sistema (como ocurría en las WSN destinadas a la detección de eventos).

2.2. Waspmites

Waspmite es una plataforma modular *opensource* que permite construir redes de sensores inalámbricas de muy bajo consumo. Su creadora es la empresa zaragozana Libelium¹, que ha conseguido que este producto tenga uno de los consumos más bajos (hasta 0,07μA) de entre todas las plataformas IoT que hay en el mercado [1].

Estos nodos equipados con sensores están compuestos de varios módulos [5]:

- ✓ Una unidad de procesamiento que consta de un microprocesador ATmega1281, capaz de trabajar a una frecuencia de 14MHz y con una memoria RAM de 8kB. Este módulo se encarga de ejecutar el código C++ del programa que ha sido compilado previamente en el IDE (*Integrated Development Environment*).



Ilustración 3. Waspmite. Fuente: www.cceiaragon.es/made-in-ceedetalle/55/waspmite

¹ www.libelium.com

- ✓ Una batería recargable a través de USB con un voltaje de hasta 4,2V que proporciona una gran autonomía al dispositivo, con una duración que puede ir desde días hasta meses según el uso. Se comercializan de manera oficial baterías de dos capacidades: 6600mA/h y 2300mA/h.

- ✓ Un módulo de comunicación para poder formar parte de la WSN junto con el resto de dispositivos. Los hay disponibles para gran cantidad de tecnologías. 802.15.4/ZigBee o Wi-Fi son dos de las más utilizadas, pero también hay módulos para 4G, Bluetooth, GPS, etc.

- ✓ Una o más placas de sensores de entre las 110 disponibles que distribuye Libelium, capaces de medir humedad, temperatura, radiación, luminosidad, etc.

Para programar los Wasmotes se utiliza un IDE que es prácticamente idéntico a la plataforma Arduino, ya que solo se diferencian en pequeños detalles del esquema de entrada y salida de datos. Sin embargo, Arduino está pensado para crear todo tipo de proyectos domésticos de forma asequible, mientras que los Wasmotes están diseñados específicamente para formar parte de redes de sensores inalámbricas [1].

Estos dispositivos disponen de una gran variedad de módulos de comunicación para acceder a multitud de tecnologías de interconexión inalámbrica. Sin embargo, cada una de ellas se ajusta mejor o peor a una situación concreta según sean las necesidades de la red que se quiere implementar.

Hay tres aspectos fundamentales a tener en cuenta a la hora de elegir una tecnología de comunicación para una red inalámbrica: la distancia entre nodos, el ancho de banda y la latencia.

- La distancia entre los miembros de una red inalámbrica puede ser desde unos pocos metros hasta varios kilómetros de distancia. Debido a que los emisores y receptores de un sistema de transmisión de datos por radiofrecuencia necesitan unas dimensiones y energía determinadas, las tecnologías inalámbricas se pueden clasificar según su alcance. Existen algunas diseñadas para conectar dispositivos móviles a otros dispositivos móviles o fijos muy próximos que se denominan redes de área personal inalámbrica (*Wireless Personal Area Networks*, WPAN). A este grupo pertenecen las tecnologías Bluetooth (con hasta 10 metros de alcance) o los enlaces infrarrojos (que necesitan estar alineados directamente) [6]. Otro tipo de tecnologías son aquellas capaces de dar cobertura a áreas más amplias, como son las redes de área metropolitana inalámbricas (*Wireless Metropolitan Area Networks*, WMAN) donde se clasifica la tecnología WiMAX (*Worldwide Interoperability for Microwave Access*), capaz de conseguir un alcance superior a los 30km [7]. Sin embargo, las redes inalámbricas con mayor alcance son las redes 3G o 4G, que se engloban en el grupo de las WWAN (*Wireless Wide Area Networks*) y que llegan a brindar cobertura a nivel nacional o global [8]. Por último, entre las redes WPAN y WMAN se encuentran las WLAN, que se han mencionado anteriormente en este trabajo y cuya cobertura puede llegar hasta los 1,5km (aunque los *routers* domésticos no suelen tener un alcance mayor de 100m) [6].

- El ancho de banda se define como la cantidad de información o de datos que se puede enviar a través de una conexión de red en un período de tiempo dado [9]. Nuevamente, las tecnologías inalámbricas pueden ser clasificadas según el valor de este parámetro de gran importancia. Algunas tecnologías como Bluetooth solo pueden llegar

a ofrecer hasta 1Mbps, pero otras como las redes Wi-Fi pueden llegar a ofrecer tasas de 600Mbps (802.11n) y coberturas de 250 metros (sin obstáculos) [10]. Es por esta razón que la tecnología Wi-Fi es la más extendida en el entorno doméstico, aunque las WSN no requieren un ancho de banda alto debido al tipo de datos que se envían.

- Por último, la latencia se refiere a la suma de retardos temporales que sufre un paquete al transmitirse a través de una red. Esto repercute en el tiempo que tarda el usuario en poder acceder a un contenido desde que accede a él hasta que lo visualiza completamente. La latencia de una red cableada suele ser menor que la de una inalámbrica, ya que estas últimas tienen un retardo mayor debido a los protocolos de control de errores que utilizan [11]. Este parámetro, pese a que es sumamente importante, no es tan determinante en el caso de una WSN ya que el objetivo principal es que los datos lleguen a su destinatario y no tienen unos requisitos estrictos de latencia.

2.3. Técnicas para la optimización de energía

Como se ha explicado en secciones anteriores, uno de los principales problemas que presentan las redes de sensores inalámbricas tiene que ver con la autonomía de sus nodos. Se hace imprescindible obtener una máxima durabilidad del sistema, y por esta razón se sigue destinando gran cantidad de tiempo a investigar formas de ahorrar energía en estos dispositivos.

En el siguiente gráfico se puede apreciar una estimación del consumo de un nodo en cada una de las diferentes etapas por las que pasa al formar parte de una WSN:

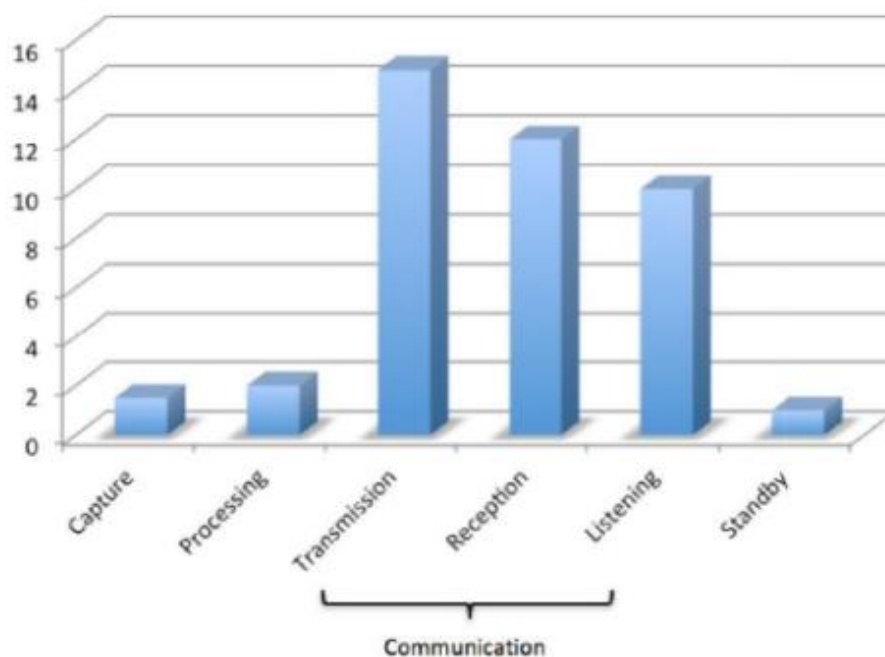


Ilustración 4. Consumo de un sensor según la etapa. Fuente: [4]

Dependiendo del contexto, las soluciones conocidas hasta el momento pueden ser clasificadas de acuerdo a tres técnicas diferentes: la administración del tiempo de funcionamiento, la estructura de los datos y la movilidad de los sensores. La elección de una u otra dependerá del tipo de WSN que se quiera configurar [4].

2.3.1. Gestión y división del tiempo

En este tipo de técnicas los nodos utilizan el modo *sleep* cuando no están realizando ninguna operación. Además, cuando no se están transmitiendo ni recibiendo datos el módulo de comunicación se mantiene apagado. Sin embargo, el uso de esta técnica requiere algún tipo de mecanismo o protocolo para encender los módulos de comunicación cuando se quiera establecer una conexión.

En los Waspmites el modo *sleep* consume únicamente 55µA frente a los 15mA que suele consumir cuando está encendido [12], aunque a este dato habría que añadir el consumo de los diferentes módulos que se le pueden acoplar, como por ejemplo el de comunicación.

Debido al notable ahorro de energía que se consigue mediante el modo *sleep*, existen distintos protocolos cuyo objetivo es optimizar el consumo de energía manteniendo un nivel aceptable de calidad en la red (paquetes perdidos, latencia de mensajes, etc.). Por ejemplo, en el protocolo GAF (*Geographical Adaptive Fidelity*) [4] la zona de cobertura se divide a su vez en varias subzonas virtuales en las que cada nodo irá turnándose cada cierto tiempo el papel de *sink node*. Esto se consigue mediante el llamado modo *Discovery*, a través del cual los nodos informan al *sink node* si quieren coger el relevo o no como único *sink node* de la subred. Así se consigue que la autonomía de éste tipo de nodo no sea mucho menor que la de sus compañeros. Además, como gran cantidad de nodos suelen estar en modo *sleep*, este protocolo consta de un mecanismo para que el *sink node*, si tiene que recibir datos de un nodo con el que no consigue establecer una buena conexión (debido a errores o interferencias), sea capaz de avisar a otros nodos intermedios para que abandonen el modo *sleep* y hagan el papel de *switch* (similar a un intermediario).

2.3.2. Técnicas orientadas a datos

Este tipo de técnicas tienen en cuenta el tipo de datos que registran los sensores. Su objetivo es ahorrar energía valorando si las mediciones son redundantes o no. Si los datos son importantes para el sistema se transmitirán al siguiente nodo, pero en caso contrario se obviarán. De esta forma el volumen de información disminuye, simplificando el procesamiento y manteniendo la precisión de todo el sistema.

El proceso de filtrado de información redundante puede haberse programado previamente (reduciendo el flujo de mediciones o adaptándolo según la situación) o puede producirse justo después de la toma de valores mediante un valor *threshold* u otro algoritmo más avanzado. Este proceso no tiene por qué ser llevado a cabo únicamente en el nodo que contiene el sensor y se basa principalmente en la predicción del comportamiento de los datos recopilados (con un razonable margen de error). Dicha predicción puede llevarse a cabo de manera aleatoria, mediante probabilidades o utilizando propiedades estocásticas, pero siempre se tendrá que evitar, en la medida de lo posible, la complejidad del algoritmo [4].

2.3.3. Técnicas basadas en la movilidad de los sensores

Las WSN compuestas por sensores móviles pueden conseguir una reducción del consumo total de energía al explorar toda la cobertura del sistema. Un conjunto de trayectorias óptimas, además de repartir el consumo entre los dispositivos, puede ayudar a evitar colapsos en la red debido a la avería de uno o más nodos.

Algunos de los modelos ya existentes proponen que los *sink nodes* de la red se vayan desplazando por unas trayectorias previamente definidas. Dicho nodo irá efectuando paradas para poder comunicarse con los sensores colindantes y recopilar información, consiguiendo ahorrar entre 5 y 10 veces más energía que las WSN de arquitectura estacionaria.

Otro ejemplo sería la llamada arquitectura *data-MULE* [4] (ilustración 5), en la que se distinguen tres niveles. El más bajo está formado por los nodos que periódicamente realizan mediciones y las almacenan en sus *buffers*. El nivel medio constaría del dispositivo *MULE*, cuya traducción al castellano significa “mula”. Este nodo móvil recorrería la red e iría recibiendo los datos almacenados en los *buffers* internos de los nodos del anterior nivel para, finalmente, pasar cerca del punto de acceso (o *gateway*) y enviar los datos al destinatario final perteneciente al nivel más alto. Este tipo de configuraciones también proporcionan nuevas posibilidades para poder cubrir grandes áreas mediante tecnologías inalámbricas con un alcance y un consumo menores. Además, también consiguen evitar pérdidas de cobertura debidas a obstáculos o a la orografía del terreno. Sin embargo, hasta que los nodos detecten un *sink node* en su vecindad deben esperar en modo activo, lo cual es bastante contraproducente desde el punto de vista del consumo energético.

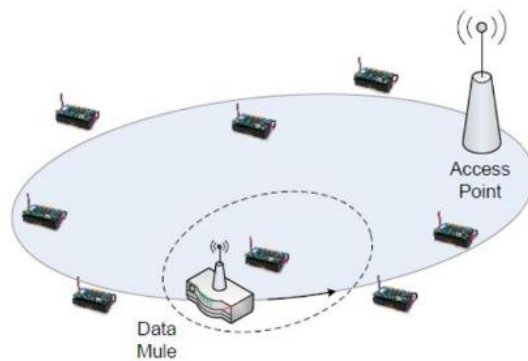


Ilustración 5. Arquitectura data-MULE. Fuente: [4]

2.4. Sincronización de hora entre nodos sensores inalámbricos

En muchas aplicaciones es útil que todos los sensores de una red estén sincronizados en hora. Existen múltiples métodos para mantener los dispositivos sincronizados, pero el más básico consiste en el intercambio de mensajes entre nodos. Este procedimiento es utilizado por el protocolo NTP (*Network Time Protocol*) [13], uno de los más utilizados en internet. En él, el nodo A le envía un mensaje al nodo B indicándole su *timestamp* (hora actual), que denominaremos t_1 . El nodo B recibe el mensaje en el instante $t_2 = t_1 + S + D_d^{(A,B)}$, donde

$D_d^{(A,B)}$ es el tiempo que tarda un paquete en llegar del nodo A al nodo B y S es el desfase entre los relojes de los dos nodos. De esta forma, el nodo B envía un mensaje de respuesta en el instante t_3 en el que incluye los *timestamps* t_2 y t_3 . Finalmente, el nodo A recibe el paquete de respuesta en el instante t_4 y estima S para poder sincronizarse con B de la siguiente forma:

$$\hat{S} = \frac{(t_2 - t_1) - (t_4 - t_3)}{2} = \frac{2S + D_d^{(A,B)} - D_d^{(B,A)}}{2} = S + \frac{u}{2}$$

Donde $u = D_d^{(A,B)} - D_d^{(B,A)}$, que en condiciones normales tendría un valor muy cercano a cero.

Otra de las opciones para la sincronización de una red inalámbrica es el esquema RBS (*Reference Broadcast Synchronization*) [14]. Dicho protocolo se basa en el envío de un *beacon packet* (que no contiene *timestamps*) por parte del *sink node* al resto de nodos. Cuando los nodos reciben este aviso inician un proceso de sincronización entre ellos similar al del protocolo NTP recientemente explicado. Este esquema elimina la incertidumbre causada por el envío del primer paquete en el protocolo NTP y, además, se supone que el *beacon packet* llega instantáneamente al resto de nodos de la red, por lo que el factor u mencionado anteriormente desaparece.

Time-synch Protocol for Sensor Networks (TSPN) [13] es otro protocolo interesante. Este esquema se extiende a través de dos fases. En la primera de ellas se establece la topología de la red y se produce la configuración inicial. A continuación, en la segunda fase, el *sink node* envía un paquete de sincronización al resto de nodos, que activan una cuenta atrás aleatoria. Cuando dicha cuenta atrás llega a su fin en cada nodo, éste inicia una comunicación de doble sentido con el *sink node* para sincronizar su reloj. Este proceso puede ir repitiéndose sucesivamente si la red dispone de más capas, es decir, más niveles en su jerarquía. Para una mejor comprensión, en la siguiente ilustración se muestra un ejemplo del esquema explicado:

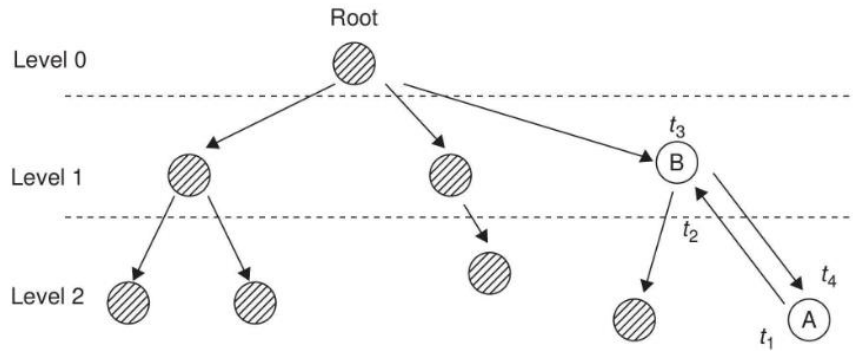


Ilustración 6. Esquema TSPN. Fuente: [13]

Otra forma de sincronizar los nodos de las WSN es el protocolo FTSP (*Flooding Time Synchronization Protocol*) [14], muy similar al protocolo TSPN pero con mejoras en algunos aspectos. FTSP utiliza cálculos de regresión lineal para minimizar el desfase entre relojes y mantener la sincronización de la red de una forma aún más precisa que otros protocolos. Además, la red se rige por una topología tipo malla en lugar de una tipo árbol, ya que el nodo raíz (*root*) va cambiando periódicamente (no siempre lo es el mismo dispositivo).

Por último, es interesante mencionar la propiedad del gradiente en relación a la sincronización de relojes. En algunas configuraciones de WSN los nodos de un mismo subgrupo interactúan entre ellos más que con el resto de nodos, por lo que es más importante mantener

una alta precisión en la sincronización entre ellos que con otros nodos más alejados. De esta forma, el máximo desfase permitido entre dos nodos formaría un gradiente que variaría según la distancia que les separa [13]. Sin embargo, esta es una de las muchas propiedades que se pueden aplicar a este tipo de protocolos definidos por multitud de parámetros a tener en cuenta.

2.5. Aplicaciones de las WSN

En secciones anteriores de este trabajo se distinguieron las aplicaciones de las redes de sensores inalámbricas en dos grupos según su finalidad. El primero de ellos era la detección de eventos y el segundo la estimación espacial y temporal de procesos. Estas redes, sea cual sea su categoría, pueden ser utilizadas en escenarios muy variados. A continuación, se explicarán cuáles son estos escenarios y se pondrán ejemplos de algunas de las aplicaciones que se han ido implementado hasta la actualidad [13].

2.5.1. Monitorización ambiental

Este tipo de aplicaciones suelen desarrollarse en el exterior, en extensiones de grandes dimensiones y durante el transcurso de varios años. Por este motivo, los dispositivos pueden llegar a estar expuestos a situaciones extremas, como terremotos o inundaciones. Esto obliga a que la red disponga de medidas de seguridad, tanto físicas como de software para evitar la pérdida o la corrupción de los datos. A su vez, dicha información debe ser accesible a tiempo real para que sea posible una reacción rápida y eficaz en el caso de que exista un peligro inminente. Los sensores tienen como objetivo detectar o monitorear algún tipo de fenómeno o parámetro natural que, con otros métodos, precisaría de la supervisión humana. Al prescindir de ella, la red debe ser robusta y capaz de asimilar errores.

Algunas de las aplicaciones más comunes son la detección de incendios o inundaciones, la medición de los niveles de contaminación del aire o la monitorización de la actividad sísmica.

Actualmente Madrid dispone de 24 estaciones de control atmosférico repartidas por toda la ciudad [15] que permiten medir los niveles de contaminación en tiempo real. Sin embargo, Madrid es el centro de un área metropolitana de grandes dimensiones y el número de estaciones a veces puede llegar a quedarse corto si se quiere obtener un mapa de mediciones más detallado. La instalación de una red de sensores inalámbricos permitiría la colocación de un número muy superior de puntos de medición sin un coste muy elevado, mejorando todo el sistema de control. Este escenario requeriría una tecnología de comunicación de largo alcance (por ejemplo, una conexión 4G) y un cifrado de los datos extremo a extremo para evitar cualquier posible intrusión.

2.5.2. Cuidados médicos

Las WSN están adquiriendo un papel importante en el mundo de la medicina por su capacidad de facilitar determinados procedimientos, tales como el rastreo de pacientes y

doctores dentro del hospital, la monitorización de datos fisiológicos humanos o la revisión y diagnóstico a distancia. Las aplicaciones de este tipo a menudo son críticas, es decir, la salud humana depende de ellas y, por tanto, la exigencia es máxima. Si una persona sufre un infarto y el sensor debe de reportar este suceso, los datos de localización deben ser muy precisos y los sensores deben estar siempre correctamente calibrados. Los nodos de estas redes rara vez son estacionarios, ya que la persona que los porta variará su posición, por lo que la movilidad de los mismos se tiene en cuenta en el diseño de la red para evitar así problemas de cobertura o de localización. Los requisitos en cuanto al tamaño del dispositivo también suelen ser muy exigentes ya que pueden ser colocados cerca de órganos vitales. Por lo general, los errores no son aceptables en este tipo de aplicaciones.

Las redes de sensores pueden ayudar, por ejemplo, en la asistencia a los pacientes (sobre todo en el turno de noche, donde hay menos personal) o en el seguimiento médico (fuera o dentro del hospital).

Un ejemplo de caso práctico, extraído de [13], se expone a continuación. Un paciente comienza a sentir un dolor en el pecho y acude al hospital para tratarlo. El doctor le diagnostica un ataque al corazón de carácter leve, lo cual no requiere cirugía, pero sí que permanezca ingresado en observación. Sin embargo, el doctor le presta un sistema de monitorización portátil que consta de una serie de sensores fisiológicos inalámbricos capaces de enviar los datos que registren al *smartphone* del paciente, desde donde una aplicación los procesará y emitirá una señal de alarma si el electrocardiograma refleja un empeoramiento de la situación. Dicha alarma será notificada al hospital, que llamará al paciente para interesarse por su estado y comprobar si es o no una falsa alarma. Este sistema permite al paciente recuperarse en su domicilio, sin necesidad de permanecer en el hospital durante varios días al cuidado del personal sanitario.

2.5.3. Posicionamiento y seguimiento de objetos o seres vivos

En las WSN los nodos pueden usarse como localizadores de gran precisión. En el caso de los animales puede ser de gran utilidad: pueden controlarse ejemplares en peligro de extinción, comprobar las rutas de emigración de las aves o detectar algún tipo de lesión o comportamiento anómalo (como, por ejemplo, una pierna rota).

La tecnología GPS (*Global Positioning System*) lleva utilizándose desde 1995 [16] para la localización de todo tipo de objetivos. A pesar de ello, se ha comprobado que su funcionamiento en interiores no es del todo preciso. Esto se debe a que el dispositivo a localizar no tiene visión directa con ninguno de los satélites del sistema GPS, lo que produce errores en los cálculos. Sin embargo, las redes de sensores inalámbricas sí pueden ser utilizadas en espacios interiores para localizar dispositivos. Esto se puede conseguir gracias al cálculo de las distancias entre los nodos que proporciona las posiciones relativas de cada uno.

2.5.4. Logística

Este campo siempre ha sido objeto de estudio con el objetivo de reducir el tiempo que tarda en llegar el producto hasta las manos del cliente, ya sea mejorando la coordinación entre las diferentes etapas por las que pasa o facilitándole el acceso al artículo a dicho cliente. Utilizando una WSN se puede conseguir que un producto sea seguido desde su producción hasta la entrega al destinatario. Esto requiere una cobertura de la red muy amplia y gran cantidad de

intermediarios involucrados en la cadena de distribución. Para las empresas distribuidoras es importante que el coste de estos sistemas no sea muy elevado y que sean capaces de lidiar con cientos o miles de productos simultáneamente.

Algunos de los casos prácticos más comunes pueden ser el seguimiento de paquetes, la localización de productos en el almacén o su gestión en el departamento de compras.

Cada día miles de pasajeros utilizan los carritos para transportar su equipaje a través de las cuatro terminales del Aeropuerto de Madrid-Barajas Adolfo Suárez. La colección de carritos que se prestan al usuario es suficiente, pero su demanda es siempre elevada y existe el problema de que haya zonas en las que el cliente no encuentre ninguno cerca porque otras personas los han ido desplazando hasta sus coches o sus terminales de facturación. Este problema se puede solucionar fácilmente con una red de sensores inalámbricos en la que cada carrito sería un nodo. Desde las oficinas del aeropuerto se podría comprobar periódicamente la localización de los carritos en un mapa y comunicarle rápidamente al personal las zonas en las que se está produciendo desabastecimiento, lo que simplificaría el trabajo de los operarios.

2.5.5. Entorno doméstico

Las posibles aplicaciones de las redes de sensores en los domicilios de los usuarios son cada vez más. Este tipo de sistemas permite el control remoto de multitud de aspectos domésticos: apagado y encendido de luces, manejo de electrodomésticos (horno, lavavajillas, lavadora, secadora, etc.), control de la calefacción o monitorización del gasto eléctrico. El concepto de hogar inteligente está cada vez más presente en las casas de nueva construcción y en apenas unos años todas estas acciones formarán parte de la vida cotidiana. Todo esto es posible gracias al ya mencionado concepto IoT (*Internet of Things*), es decir, la interconexión de cualquier producto con cualquier otro de su alrededor [17].

Un ejemplo de caso práctico puede ayudar a una mejor comprensión de las posibilidades que ofrece IoT. Una pareja acaba de comprar una casa de nueva construcción porque necesita más espacio, ya que acaban de tener un hijo. El apartamento tiene instalado una red de sensores inalámbricos capaces de controlar la iluminación, la calefacción, algunos electrodomésticos, las principales llaves de agua, etc. Un día cualquiera, justo antes de salir de casa, el hijo deja un grifo abierto sin que sus padres se percaten. La familia se monta en el coche, pero cuando se han desplazado un par de manzanas reciben una notificación en el teléfono móvil. Al haberse conectado la alarma, el sistema considera que los habitantes no están en casa y se percata de que hay un grifo que lleva tiempo encendido. Manda una notificación a través de la aplicación principal de control del hogar digital e informa al usuario de que se procede a cerrar la llave general del agua para evitar una posible inundación de la estancia. Se ha conseguido solventar el problema de forma remota. Esta es una de las infinitas posibilidades que ofrece IoT con la ayuda de las WSN.

2.5.6. Entorno industrial y comercial

Las aplicaciones de las redes de sensores en la industria tienen cierto parecido con algunas de las que se han ido mencionado hasta ahora. Una de las más comunes son los sistemas de monitorización de las cadenas de producción en las fábricas, donde los nodos miden parámetros como la temperatura, la humedad o las vibraciones de las diferentes máquinas que

trabajan en las diferentes fases del proceso de fabricación. Todos los valores pueden ser consultados por un encargado de forma remota desde el puesto de control del complejo. Si se registra algún dato anómalo el sistema genera una alarma que indica qué nodo ha registrado dicho dato y cuál es el valor límite (*threshold*) que se ha superado.

Existen otras aplicaciones que están muy relacionadas con las que se han nombrado en el campo de logística. Por ejemplo, en un supermercado se puede instalar una WSN para controlar la información de todos los productos que están a la venta: su precio, fecha de caducidad, calidad, alérgenos, etc. Para este tipo de implementaciones no se utiliza un nodo para cada producto, sino que se hace uso de las etiquetas RFID (*Radio Frequency Identification*) pasivas. Se trata de unos pequeños adhesivos que se componen de un chip y una antena y que no necesitan alimentación interna. Estas etiquetas suelen ir adheridas a los productos y para obtener la información que contienen se necesitan unos lectores que emiten señales de radiofrecuencia. Las señales emitidas por estos dispositivos llegan hasta los productos, donde los adhesivos son capaces de generar una respuesta reutilizando la energía de la señal recibida [18]. Por lo tanto, para aprovechar la tecnología RFID, los nodos de la WSN deberán ir equipados con los lectores mencionados. Todos estos elementos son capaces de integrar un sistema muy útil y de bajo coste que permite a la empresa del supermercado tener todos sus productos controlados para su venta, ya sea directamente en el establecimiento o a través de internet. Este último detalle es muy importante, ya que el comercio electrónico no para de crecer.

En la siguiente imagen se puede apreciar el aspecto de una etiqueta RFID pasiva:



Ilustración 7. Etiqueta RFID. Fuente: www.quora.com/How-do-RFID-tags-work

2.5.7. Otros campos de interés

Hay otros campos en los que una red de sensores inalámbricos puede ser de gran utilidad, como son el entretenimiento, el estado de ánimo, el transporte o el trabajo de oficina.

Existen sistemas de sensores capaces de medir el ambiente dentro de las salas de una discoteca. Esta información es muy valiosa para el DJ, ya que si los sensores detectan que hay poco ruido y movimiento en una de las salas éste puede probar con otro estilo de música o cambiar la iluminación. Si el caso es el contrario, podrá sacar partido de la animación del público y conseguir que se lleven una buena impresión del local.

Por otro lado, se han desarrollado redes de sensores inalámbricas capaces de recopilar datos que pueden describir el estado de ánimo de una persona en un momento concreto. Estos sistemas son muy parecidos a los que se utilizan en el campo de la medicina para monitorizar a

un paciente de forma remota. De hecho, una de sus aplicaciones es la monitorización de problemas mentales comunes como son las depresiones. Un estudio de los datos sobre el carácter de un ser humano a lo largo del día puede aportar pistas para animar a esa persona y mejorar su situación anímica. También puede aportar datos a tiempo real muy útiles para que las personas que le rodean entiendan mejor su situación.

En el campo del transporte, por ejemplo, se han desarrollado sistemas para el control del tráfico rodado (usando sensores de movimiento), cuya distribución es muy parecida a la del caso práctico propuesto para la medición de la contaminación en Madrid. También se podrían utilizar las WSN en otros tipos de transporte, como el tren.

Por último, las redes de sensores también pueden estar presentes en las oficinas, ayudando tanto en funciones logísticas como en funciones de localización.

2.6. Trabajos relacionados

En este punto se mencionan y se introducen brevemente algunos trabajos que tienen objetivos relacionados con el de este Trabajo de Fin de Grado, con el fin de aportar información sobre otras técnicas o puntos de vista del ahorro de energía en las redes de sensores inalámbricas.

La tesis doctoral de Jesús Fernández Bes [19] se centra en el desarrollo de dos soluciones para una gestión eficiente de las baterías de los nodos (haciendo uso principalmente de cálculos estadísticos). La primera se basa en una máquina de decisión capaz de concluir si la transmisión de un mensaje merece la pena en las condiciones energéticas actuales. Para ello no solo se tiene en cuenta el nivel de carga de la batería, sino que se tienen en cuenta más parámetros relacionados, por ejemplo, con el tipo de información a enviar o con la arquitectura de la red. La segunda solución se centra en que cada nodo debe realizar su propia estimación e ir actualizándola a partir de información recopilada por él mismo y, ahora también, por el resto de nodos. Finalmente, se testean ambas soluciones (separada y conjuntamente) y se comprueba su efectividad respecto a otras técnicas existentes. El contenido de este trabajo aporta otro enfoque para lograr extender la duración de las baterías.

Otra tesis doctoral interesante es la de M.R. Arroyo Valles [20], que explica algunas técnicas basadas en los procesos de decisión de Markov, al igual que la tesis mencionada en el párrafo anterior. Dichas soluciones son simples y se centran en determinar el valor *threshold* con el que el nodo decide si transmitir o no los paquetes de datos.

Un concepto interesante que también puede ayudar a aumentar la autonomía de los dispositivos inalámbricos es la radio cognitiva. Esta tecnología se basa en una utilización oportunista del espectro, ya que en la actualidad son muchos los estándares de comunicación que utilizan las bandas ISM y a veces se puede producir saturación espectral. En la ilustración 8 se indican los huecos libres en el espectro, respecto a los ejes de tiempo y de frecuencia, que suele aprovechar la radio cognitiva. La tesis doctoral de Elena Romero Perales [21] propone dos estrategias relacionadas con la teoría de juegos. La primera de ellas consigue ahorros de energía superiores al 65% en comparación con otras WSN no cognitivas y la segunda, más compleja (incluye el concepto de colaboración entre nodos), consigue mejorar en un 50% los datos de la

técnica anterior. De esta forma, se puede concluir que las CWSN (*Cognitive Wireless Sensor Networks*), aunque necesitan energía para la comprobación del espectro, consiguen reducir drásticamente el consumo de las baterías.

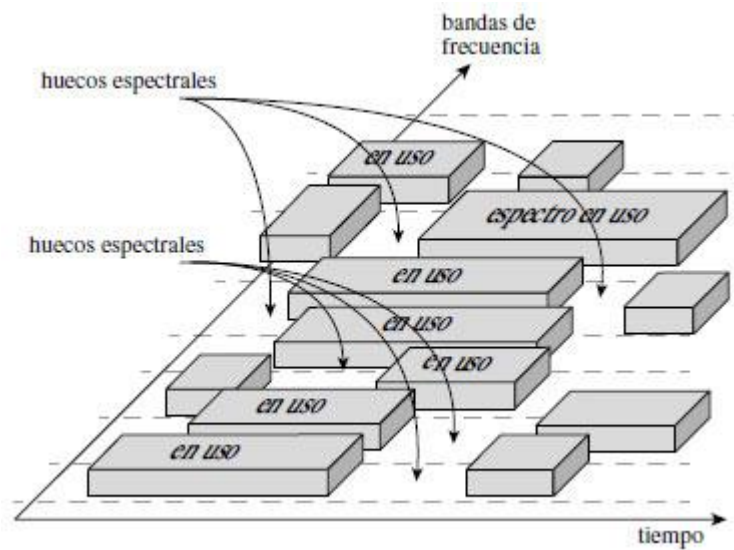


Ilustración 8. Detección de espectro libre para radio cognitiva. Fuente: www.scielo.cl/scielo.php?script=sci_arttext&pid=S0718-33052012000200007

Por último, en este trabajo se han mencionado multitud de técnicas para aumentar el ahorro de energía en las WSN. Sin embargo, Tifenn Rault menciona en su tesis [22] la posibilidad de utilizar cargadores inalámbricos que sean capaces de recargar las baterías de los nodos cuando éstas comiencen a agotarse. Otra de las ideas planteadas en esta tesis para las WSN móviles se centra en el *gateway*, concretamente en la decisión que toman el resto de nodos al transmitir o no un paquete de datos dependiendo de la distancia a la que estén de dicho dispositivo. Si la distancia supera el límite estimado en el que el paquete de datos no llegará correctamente al *Gateway*, la conexión no se llevará a cabo. De esta manera se ahorra la energía invertida en intentos fallidos.

2.7. Grandes proyectos

A continuación, se mencionan algunos proyectos de gran magnitud que han tenido, tienen o tendrán mucha importancia en el estudio, desarrollo e implantación de las redes de sensores inalámbricas en todo tipo de entornos: agricultura, industria, IoT, etc.

En el libro "*Wireless Sensor and Actuator Networks: Technologies, Analysis and Design*" [13], se menciona el proyecto europeo EYES, que tuvo una duración de tres años (desde 2002 hasta 2005) y cuyo objetivo fue desarrollar la arquitectura y la tecnología para una nueva generación de WSN que pudiese soportar gran diversidad de aplicaciones. El trabajo se enfocó principalmente en el desarrollo de nuevos esquemas de conexión, protocolos y algoritmos, todo ello en diferentes capas del modelo OSI. Se debatieron algunos ajustes del hardware de los nodos, tales como los voltajes de las baterías, los transmisores o los microprocesadores. Se realizaron pruebas con redes multi-salto y se implementaron métodos para medir la distancia

entre nodos (mediante la intensidad de la señal recibida), un parámetro que es de gran utilidad². El resultado del proyecto fue un diseño para redes de sensores inalámbricas aplicable a situaciones con una gran variedad de características: WSN móviles, diferentes tecnologías de transmisión, entornos comerciales e industriales, topologías complejas...

Por otro lado, el proyecto Glacsweb (mencionado en el mismo libro [13]) de la Universidad de Southampton (Inglaterra) tiene como objetivo monitorizar el comportamiento de los glaciares haciendo uso de redes de sensores. Con la colaboración de National Geographic, el equipo de investigación de este proyecto coloca los sensores encima y debajo de los glaciares e instala una estación base en su superficie que se encarga de recoger los datos que transmiten los nodos. A su vez, dicha estación (*sink node*) envía los datos a una estación de referencia (*gateway*) que dispone de un ordenador con conexión a internet desde el que se sube la información al servidor web. Los dispositivos son capaces de medir temperatura, presión, movimiento y otros parámetros climatológicos. Por el momento el sistema se ha instalado en dos glaciares, uno en Noruega y otro en Islandia, pero también se ha probado con él la monitorización de derrumbes en Tijuana [23]. En la siguiente ilustración se muestra un esquema sencillo de las conexiones:

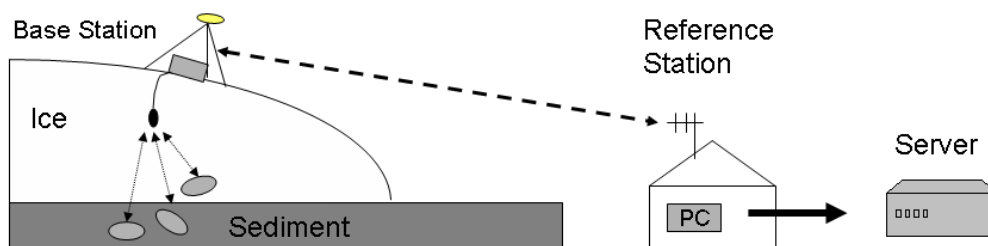


Ilustración 9. Esquema sencillo del proyecto Glacsweb. Fuente: www.glacsweb.org/technology/

Otro gran proyecto que realizó una aplicación con redes de sensores para cuidados médicos concluyó en el año 2014 y se llamaba Help4Mood [24]. Llevado a cabo por varias universidades (como la de Edimburgo o la de Valencia) y organizaciones (como i2CAT) y financiado por la Unión Europea (UE), su objetivo era desarrollar un sistema que ayudara las personas que padecen algún tipo de depresión fuerte a recuperarse en su propia casa. El proyecto se dividía en tres secciones: la primera (relacionada con este trabajo) se encargaba de monitorear al paciente a través de una WSN (con sensores en el teléfono, las llaves, el reloj o el colchón [25]) y así obtener datos de su comportamiento; la segunda se centraba en un agente virtual que preguntaba al paciente acerca de su salud y bienestar; por último, la tercera estaba enfocada a un sistema capaz de interpretar las respuestas del paciente y proporcionar datos útiles al equipo médico para facilitar su diagnóstico.

La Unión Europea financia un gran número de proyectos con el fin de fomentar el desarrollo de diversas tecnologías. El proyecto LYNCEUS [26] es uno en los que se está trabajando actualmente y su objetivo es diseñar un sistema de evacuación para hundimientos de embarcaciones utilizando una red de sensores inalámbricos. Uno de los retos más importantes es el de desplazar los nodos alrededor del barco, ya que el proyecto contempla que los sensores sean capaces de localizar a los pasajeros y comprobar su estado básico de salud para ayudar a las unidades de emergencia a salvar el mayor número de vidas posibles. Además, el sistema también incluiría el uso de drones para la localización de pasajeros en el agua y la

² Tal y como se ha explicado en la sección anterior.

identificación de los mismos utilizando el número de identificación de los chalecos salvavidas. Todo ello teniendo en cuenta los protocolos ya diseñados para este tipo de situaciones.

Por último, otro proyecto financiado por la UE y llevado a cabo por una asociación de universidades y empresas de Oporto (Portugal) es FUTURE-CITIES [27]. El objetivo del proyecto, que comenzó en 2012, es convertir esta ciudad portuguesa en un laboratorio para ciencias y tecnologías urbanas para las denominadas *smart cities* (ciudades inteligentes). Actualmente ya existen infraestructuras distribuidas por la ciudad que se agrupan en tres categorías: la primera se denomina VANET (*Vehicular Ad-hoc Networking*) y está compuesta por una red de vehículos interconectados que se localizan en el entorno urbano y en el puerto; la segunda se llama UrbanSense Platform y es una gran infraestructura de sensores inalámbricos capaces de monitorear diversos parámetros a través de sensores ambientales y contadores de peatones; por último, la tercera categoría, cuyo nombre es SenseMyCity, engloba toda la infraestructura necesaria para aglutinar todos los datos y presentarlos de forma simplificada en una aplicación para *smartphones*. Todo el sistema consta con más de 800 nodos, estáticos o móviles, que se conectan a internet a través de los puntos de acceso Wi-Fi distribuidos por toda la ciudad pertenecientes a la asociación Porto Digital, aunque también hay algunos que utilizan una conexión por cable de fibra óptica. Este proyecto (también conocido como Porto Living Lab) es una buena fuente de información para muchos estudios relacionados con la sostenibilidad y la organización de las grandes ciudades y, por esta razón, multitud de universidades de toda Europa y gran cantidad de empresas están interesadas.

2.8. Marco regulador

2.8.1. Estándares técnicos

En este punto se nombran y explican brevemente los estándares técnicos que están siendo más utilizados actualmente en las diferentes aplicaciones de las redes de sensores inalámbricas con el fin de garantizar la interoperabilidad de equipos de distintos fabricantes.

- IEEE 802.11 para Redes Inalámbricas de Área Local (Wi-Fi):

La especificación 802.11 es un estándar internacional que define las características de una WLAN (Wireless Local Area Network). Por el uso indebido de los términos (y por razones de marketing) el nombre del estándar se confunde con el nombre de la certificación Wi-Fi, otorgada por la Wi-Fi Alliance, que garantiza la compatibilidad entre dispositivos que utilizan el IEEE 802.11 [28].

Esta tecnología suele tener un alcance de unos 60-90 metros (en equipos domésticos) y una velocidad máxima de 6,93Gbps (aunque a medida que el usuario se aleja del router, ésta disminuye drásticamente dependiendo de los obstáculos a los que se enfrente la señal). Existen varias versiones (enmiendas), pero son tres las más utilizadas: 802.11a, 802.11b o 802.11g. Más tarde han aparecido otras versiones como 802.11n y 802.11ac que mejoran notablemente las prestaciones de sus dos predecesoras y que actualmente están muy implantadas. La mayoría de dispositivos actuales ya son capaces

de conectarse al estándar 802.11n, pero los *routers* suelen utilizar varias versiones para evitar posibles incompatibilidades [29].

En la siguiente tabla se puede observar una breve comparación entre las características técnicas más significativas de las cinco últimas versiones mencionadas:

	802.11a	802.11b	802.11g	802.11n	802.11ac
Banda de frecuencia	5GHz	2,4GHz	2,4GHz	2,4/5GHz	5GHz
Espectro disponible	20MHz	20MHz	20MHz	40MHz	160MHz
Máxima tasa binaria	54Mbps	11Mbps	54Mbps	600Mbps	1.300Mbps
Alcance máximo (outdoor)	120m	140m	140m	250m	~150m
Año de publicación	1999	1999	2003	2009	2013

Tabla 2. Comparativa versiones IEEE 802.11. Fuentes: [10] [28] [29]

Este estándar establece los protocolos necesarios para la conexión inalámbrica en las capas física y de enlace (LLC y MAC). En la capa física se define la modulación utilizada y las características de señalización para la transmisión de datos, mientras que en la capa de enlace se define la interfaz entre el bus del dispositivo y la capa física (mediante un método de acceso parecido al que se utiliza en el estándar Ethernet) y las reglas para la comunicación entre nodos.

Las principales ventajas de este estándar son su bajo coste y su cobertura (amplia y difícil de obstruir), mientras que sus desventajas más importantes son las interferencias en la banda de 2,4GHz (con hornos microondas, dispositivos BlueTooth, teléfonos inalámbricos, etc.) y las posibles incompatibilidades mencionadas según la versión que se utilice [30].

- Estándar IEEE 802.15.4 para Redes Inalámbricas de Área Local o Metropolitana:

En este estándar se describe el protocolo compatible para la interconexión de dispositivos de comunicación de datos por radiofrecuencia de baja velocidad, bajo consumo energético y corto alcance en redes inalámbricas de área personal (*Wireless Personal Area Networks*, WPAN). Este estándar fue publicado en el año 2003 y revisado en 2006 y 2011 [31], y es el estándar más utilizado en las WSN. Otros aspectos que definen a las redes que lo utilizan son la flexibilidad de la red y el bajo coste.

El proyecto IEEE 802 divide la capa de enlace de datos (Data Link Layer, DLL) en dos subcapas: la subcapa de enlace de acceso a medios (Medium Access Control, MAC) y la de control de enlaces lógicos (Logical Link Control, LLC). El LLC es común a todos los estándares 802, pero la subcapa MAC depende del hardware implementado. El MAC de IEEE 802.15.4 utiliza un protocolo basado en el algoritmo CSMA/CA, el cual requiere escuchar el canal antes de transmitir para evitar posibles colisiones con otras transmisiones [3]. Además, la capa MAC también permite el uso de una variante llamada CSMA/CA ranurado, en la que se utilizan pequeños períodos de tiempo (desde 15ms) para organizar las transmisiones (además de la comprobación del canal) [32].

Las principales características técnicas que definen el estándar IEEE 802.15.4 se resumen en la tabla 3 (de la siguiente página).

Propiedad	Rango de valores
Bandas de frecuencia y tasa binaria máxima	868MHz (Europa) → 20kbps 915MHz (EEUU) → 40kbps 2,4GHz → 250kbps
Alcance	10-20m
Latencia	Inferior a 15ms
Canales	868MHz → 1 canal 915MHz → 10 canales 2,4GHz → 16 canales
Direccionamiento	Mín. 8 bits – Máx. 64 bits
Canal de acceso	CSMA/CA y CSMA/CA ranurado

Tabla 3. Propiedades IEEE 802.15.4. Fuente: [32]

Respecto a las topologías utilizadas, el documento permite dos tipos (aunque no están totalmente definidas ya que el estándar solo se centra en las capas física y MAC): en estrella o peer-to-peer. Esta última topología se representa esquemáticamente en la ilustración 10.

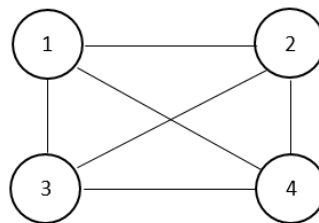


Ilustración 10. Topología peer-to-peer.

Sin embargo, aunque el IEEE 802.15.4 solo está definido entorno a dos capas, existe una tecnología que define las capas de red y aplicación llamada ZigBee [3]. Promovida por la Zigbee Alliance (un ecosistema internacional de compañías como Motorola, Philips, Samsung o Siemens), se caracteriza por operar redes de gran densidad y por permitir la existencia de varias redes en el mismo canal (gracias a los identificadores únicos de red). Además, es un protocolo de comunicación multi-salto (un mensaje puede llegar de un nodo a otro que esté fuera de alcance si existen nodos intermedios) con topología tipo malla (ilustración 11). Respecto a la capa de red, ZigBee brinda las funciones necesarias para iniciar la red, añadir nodos y enrutar y manejar paquetes (basándose en el protocolo *Ad Hoc On-Demand Vector Routing*, AODV) en la WSN [33].

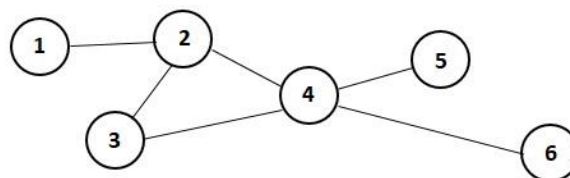


Ilustración 11. Topología tipo malla.

Otra tecnología que aprovecha las características de las dos capas del estándar IEEE 802.15.4 es la denominada 6LoWPAN (IPv6 over Low power Wireless Personal Area Networks), que posibilita el uso de IPv6. Esto permite a los nodos comunicarse directamente con otros dispositivos IP [3].

Existen más tecnologías que aprovechan las capas física y MAC de este modelo como pueden ser WirelessHART o ISA.100.11a, que adaptan dispositivos utilizados en el entorno industrial a la tecnología inalámbrica [34].

IEEE 802.15.4 es el estándar más relevante de entre todos los que se utilizan para controlar las redes de sensores inalámbricas, ya que puede conseguir grandes coberturas (mediante comunicación multi-salto) y es económica. En la actualidad se han añadido multitud de enmiendas que van desde la 802.15.4a hasta la 802.15.4u.

- IEEE 802.15.1 para Redes Inalámbricas de Área Local o Metropolitana:

Este estándar también pertenece a la familia 802 y en él se definen las capas inferiores (física y MAC) para la tecnología Bluetooth. También se especifica una cláusula para los puntos de acceso al servicio (*Service Access Point*, SAP), que incluye una interfaz LLC-MAC, y una normativa que proporciona un borrador para un protocolo PICS (*Protocol Implementation Conformance Statement*) [35].

La capa física de Bluetooth utiliza la banda de frecuencias ISM de 2,4GHz con 79 canales y una velocidad máxima de 1Mbps. Los nodos se organizan en piconets, donde un nodo maestro puede tener un máximo de siete nodos esclavos activos. El canal físico se divide en slots de 625µs de duración, consiguiendo una conexión full-duplex [3].

La principal limitación de Bluetooth para su aplicación en una WSN es su corto alcance, que (como se ha mencionado en secciones anteriores) se extiende hasta los 10 metros. Aun así, se dan casos en los que se elige este tipo de conexión ya que es una tecnología robusta, de bajo consumo y que conlleva un pequeño gasto económico.

Además de los estándares mencionados, existen muchos otros para adaptar las capas superiores del modelo OSI a diversos dispositivos, protocolos o tecnologías, como pueden ser los elaborados por las organizaciones EPCglobal (para RFID), W3C (para *eXtensible Markup Language*, XML), IETF (para configuraciones de seguridad) [34], etc.

Con relación a los Waspnotes, actualmente Libelium está promocionando el uso de una nueva tecnología de comunicación de baja potencia y largo alcance llamada LoRaWAN (*Low Power Wide Area Network*), que está teniendo gran difusión en Europa, Asia y Norteamérica. Este estándar, publicado por LoRa Alliance, es capaz de conectar dispositivos separados a kilómetros de distancia, operar en tres bandas distintas de frecuencia y proteger los datos mediante autenticación [36].

2.8.2. Análisis de la legislación aplicable a la solución implementada

Los estándares publicados por el Instituto de Ingenieros Eléctricos y Electrónicos (*Institute of Electrical and Electronics Engineers*, IEEE) pueden ser utilizados libremente, ya que se trata de una organización sin ánimo de lucro. Los profesionales y estudiantes que integran la asociación también se ocupan de actualizar dichos documentos para adecuarlos a los cambios que se suceden continuamente en el entorno tecnológico [37]. Además, todas las tecnologías de comunicación mencionadas hasta ahora utilizan un rango de frecuencias (bandas ISM,

Industrial, Scientific and Medical) que no requiere ningún tipo de licencia para su uso. Esto se debe a que la forma en la que están definidos los estándares ya garantiza no se interfiere con usuarios primarios de las bandas correspondientes. Por lo tanto, para el diseño de una red de sensores inalámbricos no se necesita ningún tipo de permiso en lo que se refiere a este aspecto.

A menudo, en algunas empresas, las WSN pueden llegar a transmitir datos de carácter personal, es decir, cualquier información numérica, alfabética, gráfica, fotográfica, acústica o de cualquier otro tipo concerniente a personas físicas identificadas o identificables [38]. La manipulación de este tipo de información debe cumplir la Ley Orgánica de Protección de Datos (LOPD), de la que es responsable la Agencia Española de Protección de Datos. Dicho documento obliga a cumplir una serie de requisitos y determinadas medidas de seguridad y establece un régimen de sanciones para las infracciones que se cometan [39].

A raíz de lo que se acaba de explicar, dependiendo de la sensibilidad de los datos que se manipulen en una red de sensores inalámbricos se adoptarán unas u otras medidas de seguridad. La más común es el cifrado de los datos, ya que evita que la información sea legible si es interceptada por un usuario conectado a la red. Otra medida para la protección de datos se basaría en no permitir unirse a la red a todos o a algunos usuarios, es decir, a controlar el acceso a la red (ya sea por contraseña o de otra forma). Esta última medida, además, protege la red de cualquier acción malintencionada que pudiese alterar su correcto funcionamiento. Es importante mencionar que las redes son especialmente vulnerables a los virus, ya que éstos son capaces de extenderse rápidamente por ella y contagiar al resto de nodos.

Aunque se apliquen importantes medidas de seguridad, una red siempre está expuesta al riesgo de una intrusión. Por esta razón, si se quiere evitar la pérdida de información se deben establecer mecanismos para la recuperación de los mismos. En algunas redes se suele realizar de forma periódica una copia de seguridad de los datos, mientras que en otras ocasiones la información se va trasladando a un archivo definitivo cuando ya no va a volver a ser modificada. Por último, si la situación lo requiere, se podría disponer de otra red y otro servidor alternativos que evitasen grandes períodos de tiempo de inoperatividad en el sistema si se produjese un desastre (como, por ejemplo, un incendio) [40].

3. DISEÑO DE LA SOLUCIÓN PROPUESTA

En este capítulo se procederá a explicar detalladamente el diseño y el funcionamiento de la solución síncrona que se propone en este proyecto, junto con multitud de aspectos que se han tenido en cuenta para ello.

3.1. Descripción del diseño propuesto

Teniendo en cuenta los objetivos iniciales mencionados en el capítulo 1 de este documento, el diseño del sistema elaborado en este trabajo tiene como fin conseguir un ahorro de batería apagando el módulo de comunicación. Para ello se ha elegido hacer uso de una de las funcionalidades que tiene la plataforma Wasmote: el modo *sleep*. Al activar esta modalidad, el nodo desactiva todos los módulos que están conectados a él salvo el módulo del RTC (*Real Time Clock*), para poder detectar la interrupción generada por una alarma previamente configurada. Los nodos del sistema deben mantener la capacidad de iniciar y concluir conexiones con otros dispositivos de la red como si se tratase de una solución asíncrona. Para conseguirlo, el diseño propuesto debe ser capaz de programar períodos de intercambio de datos utilizando el sincronismo entre nodos.

A continuación, se muestra un esquema simplificado de la solución propuesta:

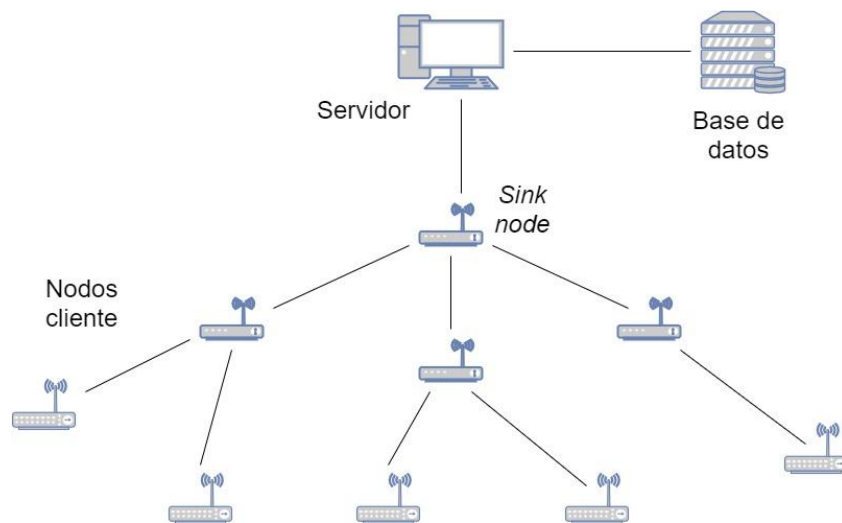


Ilustración 12. Esquema de la solución propuesta³.

Como se puede observar en la imagen, el *sink node* es el encargado de enviar las mediciones al servidor, que a su vez las podrá guardar en una base de datos, formando una topología tipo árbol. El flujo de información queda establecido en la ilustración en sentido ascendente, es decir, los datos se transmiten desde los nodos cliente hasta la base de datos pasando primero por el *sink node* y luego por el servidor.

Este diseño está destinado a redes de sensores inalámbricas que tengan un excesivo consumo energético y necesiten una solución sencilla que lo disminuya drásticamente. Dichas

³ Esta figura se ha elaborado haciendo uso de la herramienta www.draw.io

redes se podrán beneficiar de la solución propuesta en este trabajo adaptando su sistema de medición a las conexiones periódicas entre nodos que se utilizan para el intercambio de datos. Además, la WSN podrá ser configurada rápidamente fijando los parámetros a manejar por el mecanismo de configuración que incluye el diseño propuesto.

3.2. Requisitos generales

En primer lugar, antes del comienzo del proceso de diseño se establecieron los requisitos que la WSN debía cumplir en cualquiera de sus implementaciones. Dichas obligaciones se enumeran a continuación:

- ✓ Los nodos que conforman la red de sensores deben ser capaces de comunicarse bidireccionalmente, es decir, deben soportar conexiones Wasmote-servidor y Wasmote-Wasmote. Para ello podrán utilizar una red de infraestructura o una conexión tipo *ad-hoc*⁴.
- ✓ El sistema debe de tener un mecanismo mediante el cual, desde el servidor, se puedan cambiar uno o más ajustes de configuración básicos de la red sin modificar el código C++ implementado en los Wasmotes.
- ✓ Las mediciones realizadas por los sensores se podrán ir almacenando de forma progresiva en una base de datos que debe ser accesible desde el servidor y desde donde se podrá consultar la información en cualquier momento.
- ✓ La red debe de funcionar correctamente para cualquier número de nodos cliente⁵.

3.3. Descripción del sistema implementado

Con el fin de testear el diseño propuesto (tal y como se indica en el objetivo principal), se ha implementado un prototipo de WSN basado en dicha solución.

3.3.1. Hardware y software utilizado

El material (hardware) utilizado en la implementación del prototipo se especifica a continuación:

⁴ Este requisito no se ha cumplido en la solución final implementada debido a motivos que se explicarán más adelante.

⁵ Aunque se deberán añadir más *sink nodes* a la red si el número de nodos cliente asociados a un *sink node* es alto con el fin de mejorar el rendimiento del sistema.

- ✓ Tres sensores inalámbricos autónomos (concretamente Waspmotes PRO v1.2 con baterías recargables de 6600mA/h).
- ✓ Un módulo de comunicación Wi-Fi por cada sensor utilizado.
- ✓ Un cable que permite conectar el dispositivo inalámbrico (sensor) al ordenador (cable USB Macho a Mini 5 USB Macho).
- ✓ Un *router* Wi-Fi con acceso a internet (modelo Askey RTF3505VW).
- ✓ Un *host* conectado a internet capaz de actuar como servidor web y que tiene acceso a una base de datos (se ha contratado un servicio de *hosting* con la empresa Hostinger⁶ que también incluye la creación de una base de datos).

No se han utilizado módulos específicos para la medición de magnitudes en los Waspmotes debido a que el propósito de este trabajo afecta directamente al diseño y configuración de la red y no al tipo de mediciones que se realizan.

Es importante mencionar la razón por la que se ha elegido utilizar en los sensores módulos de comunicación Wi-Fi en detrimento de todas las demás tecnologías inalámbricas disponibles. Existen otras alternativas (como ZigBee o BlueTooth) que consiguen un menor consumo de energía. No obstante, la principal ventaja de la tecnología Wi-Fi, que ha propiciado su elección, es su gran flexibilidad: tiene un rango de cobertura mayor y cuando un dispositivo detecta un AP (*Access Point*) es capaz de conectarse a internet sin tener que usar *gateways* especializados. Esta característica proporciona al usuario una mayor variedad de posibles implementaciones utilizando la solución sincronizada propuesta.

Para poder utilizar todo el material que se acaba de enumerarse necesita un software que le aporte funcionalidad. En esta categoría se incluyen: los drivers necesarios para la conexión alámbrica entre los sensores y el ordenador; el IDE (*Integrated Development Environment*), que compila y carga los archivos de código C++ en los Waspmotes; y, por último, el panel de control del servidor web, que permite subir los archivos PHP y manejar la base de datos (mediante phpMyAdmin).

3.3.2. Esquema de conexiones

Como se ha explicado en el capítulo 2, una de las topologías más utilizadas en las redes inalámbricas de sensores es la topología en estrella, que es la que se ha implementado en esta solución. Esta arquitectura se compone de un *sink node* que recibe todos los mensajes directamente (sin intermediarios) del resto de nodos de la red, por lo que se trata de un sistema sencillo y, como consecuencia, robusto.

Sin embargo, aunque los nodos de la WSN siguen la topología que se acaba de mencionar, existen otros dos dispositivos que también están incluidos en el sistema: el *gateway* y el servidor web. El primero de ellos es un *router* que proporciona al *sink node* acceso a internet con la creación de una red de infraestructura Wi-Fi. El segundo es un servidor web que contiene archivos de código PHP 5.6 capaces de interactuar por un lado con la base de datos mediante

⁶ www.hostinger.es

comandos SQL (*Structured Query Language*) y por otro con el *sink node* (que ahora actuaría como cliente) respondiendo a sus peticiones GET (uno de los dos métodos HTTP más conocidos).

En la siguiente imagen se puede observar un esquema sencillo de la estructura de la red implementada junto con una flecha que indica la dirección y el sentido del flujo de datos:

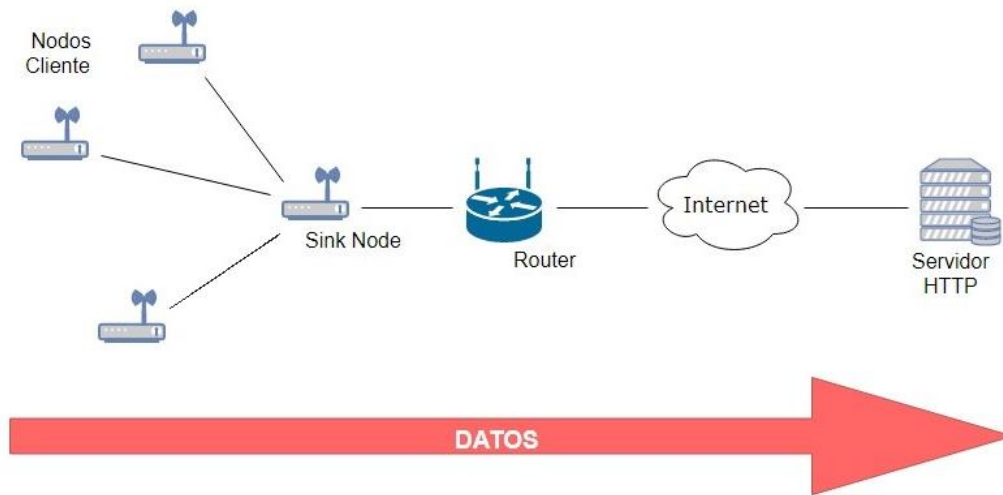


Ilustración 13. Esquema sencillo del prototipo implementado⁷.

Como se puede apreciar, el esquema del sistema completo no tiene una topología en estrella, sino que sigue una arquitectura de red tipo árbol (ya que el *sink node* no actúa como nodo central). Por esta razón, es importante distinguir que la red de sensores inalámbricos sigue una topología en estrella, pero si se tiene en cuenta toda la solución que se ha implementado la arquitectura final es de tipo árbol.

El esquema de la solución propuesta en este proyecto (apartado 3.1.) hace uso de conexiones Wasmote-Wasmote. Sin embargo, en pleno desarrollo del prototipo no se ha conseguido llegar conectar dos Wasmotes directamente a nivel 2. No existe problema a la hora de establecer una conexión *ad-hoc* entre un dispositivo capaz de crear una red de este tipo (como puede ser un ordenador o un *smartphone*) y un Wasmote, pero al intentarlo con dos Wasmotes existe algún tipo de problema relacionado con el reconocimiento de los nodos o con la identificación de la red que no se ha podido llegar a solventar. Debido a ello, para conseguir transmitir los datos desde los nodos cliente hasta el *sink node*, se decidió que los primeros también estuviesen conectados a la red de infraestructura creada por el *router* y que estableciesen una conexión TCP cliente-servidor con el *sink node*. Con este cambio todos los nodos de la WSN tendrían acceso a internet y podrían comunicarse directamente con el servidor HTTP, pero con el objetivo de que en un futuro se consiga solucionar el problema se condujo que los nodos cliente se comportasen como si no tuviesen acceso a internet (simulando no utilizar el AP). Se decidió así no dedicar más tiempo al funcionamiento del modo *ad-hoc* en los Wasmotes, pues el cambio a modo infraestructura no afecta a la validación de la propuesta inicial (a nivel 3 la comunicación es Wasmote-Wasmote).

Finalmente, después del cambio que se acaba de explicar, el prototipo implementado se puede apreciar a continuación (en la siguiente página), donde solo el *sink node* (servidor) utiliza el *gateway (router)* para comunicarse con el servidor HTTP:

⁷ Esta figura se ha elaborado haciendo uso de la herramienta www.draw.io

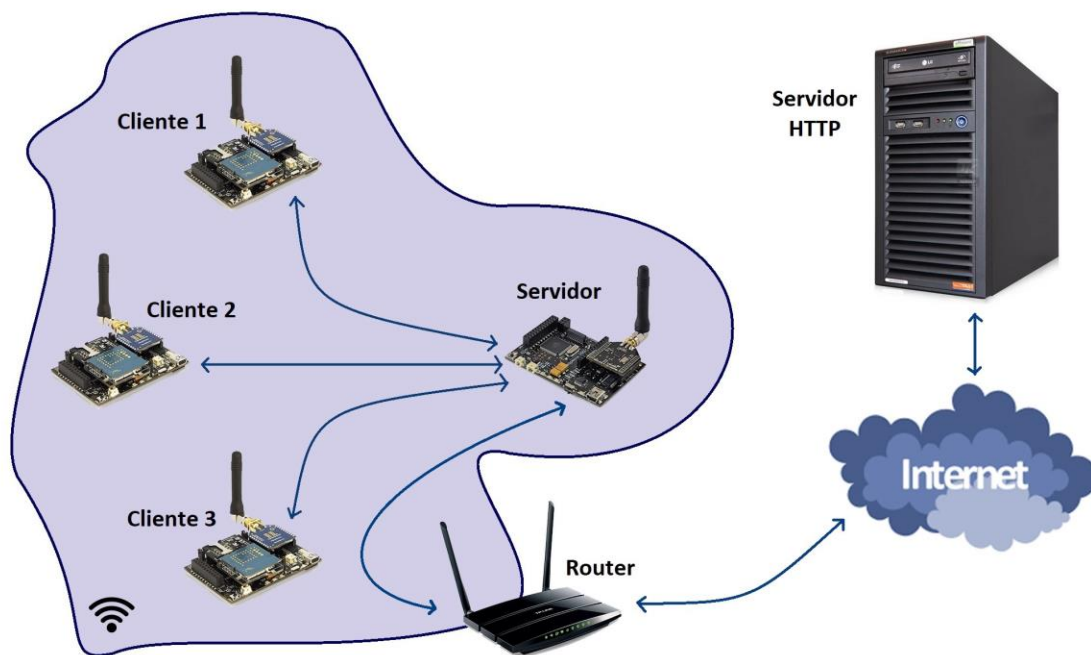


Ilustración 14. Prototipo implementado.

Uno de los inconvenientes que tiene esta configuración es que los nodos clientes tienen que estar dentro del rango de cobertura del *router*, mientras que antes solo necesitaban estar lo suficientemente cerca del *sink node* para poder llegar a establecer una conexión tipo *ad-hoc* con él.

3.4. Desarrollo del sistema implementado

En esta sección se describen y se justifican todos los pasos que se han llevado a cabo hasta llegar a conseguir el correcto funcionamiento del prototipo basado en el diseño propuesto.

3.4.1. Red de sensores inalámbricos (WSN)

En primer lugar, los esfuerzos se centraron en la implementación de la WSN con topología en estrella. Se realizaron los ajustes necesarios para que dos Wasmotes se conectasen a la red Wi-Fi gestionada por el *router* y estableciesen una conexión TCP cliente-servidor. Durante las pruebas se detectó que algunos módulos Wi-Fi de los Wasmotes nunca se llegaban a encender o no funcionaban correctamente (por ejemplo, no eran capaces de conectarse a un punto de acceso cualquiera). Dichos módulos Wi-Fi fueron reemplazados y, aunque hubo algún otro módulo más adelante que también causó problemas, no entorpecieron en exceso el desarrollo del sistema. Sin embargo, es importante destacar que estos módulos se pueden deteriorar y pueden llegar en algunos casos a funcionar de forma deficiente o a perder su funcionalidad por completo. Cuando se consiguió establecer la conexión TCP entre dos Wasmotes se pasó a definir la estructura de los mensajes que se utilizarían para transportar

los datos de las mediciones. Debido a la simplicidad de los mensajes se optó por enviar cadenas de caracteres sobre TCP como la que se muestra a continuación:

Nombre&2-> *Mensaje1*/00-> *Mensaje2*/00

Primero se indica el nombre⁸ del nodo origen seguido del símbolo ‘&’ y el número de mensajes y, por último, se incluyen los mensajes entre un símbolo ‘->’ que marca el inicio y otro ‘/00’ que marca el final. De esta forma el *sink node* identifica al nodo que ha originado el mensaje y se prepara para guardar el número de mensajes especificado⁹.

En este punto del desarrollo se pudo comenzar a sincronizar todos los nodos para poder exprimir las ventajas del modo *sleep* que se explicaron en el capítulo 2. Por simplicidad se procedió a fijar la misma hora en el RTC de cada sensor como primer paso en el código C++ que llevan cargado. De esta forma, al encender todos los Wasmotes a la vez éstos quedan sincronizados y se pueden establecer alarmas para que, periódicamente, el *sink node* vaya recibiendo los datos que le envían los nodos cliente. El RTC permite utilizar dos alarmas simultáneamente (aunque tienen distinta precisión) capaces de generar un tipo de interrupción [41], por lo que los Wasmotes utilizan para ‘despertarse’ la alarma 1, que se vuelve a configurar una vez se produce dicha interrupción de forma periódica. Para el prototipo implementado se determinó que los nodos clientes deberían enviar datos al *sink node* una vez por cada hora natural transcurrida. Esta decisión originó el concepto de minutos de conexión (que se utilizará a partir de ahora en este documento) que se refiere al minuto exacto de la hora configurada en el RTC de los nodos en el que se produce una de las conexiones periódicas *sink node*-nodo cliente¹⁰.

Aunque toda la WSN ya funcionaba correctamente, se decidió añadir una mejora para la recuperación de mediciones perdidas. Como algunas veces los módulos Wi-Fi se pueden deteriorar y fallar a la hora de encenderse o conectarse a la red (aunque también puede haber otras causas de fallos como interferencias o problemas con el AP), se implementó un sencillo mecanismo que permite al nodo cliente guardar un mensaje del que no se tiene confirmación de que haya llegado al *sink node* para poder enviarlo junto con un nuevo mensaje en la siguiente conexión. Por eso se indica en el paquete de datos el número de mensajes y también por esta razón el *sink node* está preparado para almacenar temporalmente dos mensajes y enviarlos ordenadamente al servidor HTTP.

3.4.2. Servidor HTTP

En segundo lugar, el desarrollo del sistema se centró en la configuración del servidor HTTP para que éste se pudiera comunicar con la WSN implementada para enviarle parámetros de configuración y para darle acceso a la base de datos. Se procedió a contratar un *host* de pago en Hostinger (para ahorrarse los problemas de disponibilidad del servidor que suelen aparecer en las subscripciones gratuitas) que incluía la posibilidad de crear una base de datos en un

⁸ Para la solución implementada los nombres de los nodos tienen una longitud de 4 caracteres.

⁹ La razón por la que un mismo paquete de datos puede transportar dos mensajes se explica en el último párrafo del punto.

¹⁰ Este esquema se eligió por su sencillez, pero se podría generalizar fácilmente. Por ejemplo, los minutos de conexión podrían indicar el período entre conexiones de un nodo y conseguir así una mayor flexibilidad en la programación de las conexiones.

dominio de la empresa dedicado a ello y manejarla a través de la herramienta phpMyAdmin. A continuación, se creó una base de datos MySQL que contiene dos tablas:

- La tabla “config”:

En ella se guardan algunos parámetros de configuración básicos de los nodos que componen la WSN, de tal forma que cada fila se corresponde con un dispositivo. Está compuesta por cinco columnas:

id	De tipo <i>integer</i> , es la clave primaria de la tabla, es decir, actúa como identificador. Su valor para cada fila nueva se incrementa en una unidad respecto a la fila anterior (auto-incrementable).
nombre	De tipo <i>text</i> , este campo contiene el nombre del nodo. Se utiliza como identificador en las consultas que se realizan desde los archivos PHP.
admin	De tipo <i>text</i> , contiene el valor “si” si se trata del <i>sink node</i> o el valor “no” si se trata de un nodo cliente.
ip	De tipo <i>text</i> , contiene la dirección IP asociada al nodo. ¹¹
minConex	De tipo <i>integer</i> , indica en qué minuto debe de saltar la interrupción de la alarma del RTC, es decir, determina en qué momento exacto se produce la conexión periódica (una vez por hora) con el <i>sink node</i> . ¹² Su valor mínimo es 0 y su valor máximo es 59. En el caso del <i>sink node</i> este campo no se utiliza y su valor es irrelevante.

Tabla 4. Tabla “config” de la base de datos.

- La tabla “data”:

En ella se almacenan todas las mediciones de los sensores que recibe el servidor HTTP enviados por el *sink node*. Cada mensaje que se almacena en la base de datos está representado por una fila. La tabla está compuesta también por cinco columnas:

id	De tipo <i>integer</i> , sus características son las mismas que las de la columna que tiene su mismo nombre en la tabla “config”.
nombre	De tipo <i>text</i> , este campo contiene el nombre del sensor que ha realizado la medición.
datos	De tipo <i>text</i> , es la columna más importante ya que contiene la información recopilada por el/los sensores del Waspote.
hora	De tipo <i>text</i> , contiene la hora en la que se añadió la fila a la base de datos (hora local del servidor HTTP).
fecha	De tipo <i>text</i> , indica la fecha en la que se añadió la fila a la base de datos (hora local del servidor HTTP).

Tabla 5. Tabla “data” de la base de datos.

¹¹ Este campo no se utiliza en la solución final ya que las direcciones IP son fijas y se asignan al cargar el código en cada uno de los Waspotes. Sin embargo, durante el desarrollo del sistema se llegó a usar para asignar las direcciones IP desde el servidor HTTP: los nodos se conectaban a la red Wi-Fi con una IP asignada automáticamente por el servidor DHCP (*Dynamic Host Configuration Protocol*) y descargaban y cambiaban la dirección IP (haciéndola fija) para utilizarla en las siguientes conexiones.

¹² Un detalle a tener en cuenta es que nunca puede haber dos valores “minConex” iguales para nodos cliente ya que se podrían producir errores en la conexión TCP y, además, el *sink node* solo espera un mensaje cada vez que salta su alarma.

Una vez creada la base de datos el siguiente paso fue configurar la comunicación entre el *sink node* y el servidor HTTP. Se elaboraron dos archivos PHP (cuyo código completo se adjunta en el anexo III) capaces de responder a las peticiones GET enviadas por el *sink node*:

1. “config.php” (URL: www.carlinhopm.esy.es/config.php):

Se encarga de comprobar que el nodo que efectúa la petición efectivamente es el *sink node* y, acto seguido, le responde con un mensaje que contiene el nombre y el minuto de conexión de todos y cada uno de los nodos clientes que están registrados en la tabla “config” de la base de datos (ordenados según su minuto de conexión). Un ejemplo de la petición GET y otro de la respuesta generada por el archivo PHP, respectivamente, se detallan a continuación:

&nombre = COM6

&nombre = COM4&minConex = 4&nombre = COM3&minConex = 34

En la petición GET el *sink node* simplemente le proporciona su nombre al servidor HTTP (en el ejemplo “COM6”), mientras que la respuesta de éste último puede ser de mayor o menor longitud dependiendo de los nodos clientes que se hayan registrado en la tabla “config” de la base de datos. En la respuesta del servidor HTTP se especifica el nombre (en el ejemplo “COM4” y “COM3”) y el minuto de conexión (“4” y “34”) de cada nodo cliente.

2. “dataServer.php” (URL: www.carlinhopm.esy.es/dataServer.php):

Este archivo se ocupa de recibir las mediciones y almacenarlas en la base de datos. Recibe una petición GET del *sink node* como la que se muestra a continuación:

nombre = COM4&datos = XXXXXX

Como se puede observar, en la petición se especifica el nombre del sensor que ha realizado la medición (“COM4”) y los datos recopilados. El archivo se encarga de agregar una fila nueva en la tabla “data” con esta información a la que también añade la fecha y la hora para rellenar todas las columnas en la base de datos. Por último, responde al *sink node* con el siguiente mensaje (que no varía en ningún caso):

&UPDATED – Datos actualizados

Ambos archivos también disponen de mensajes de error que se le envían al *sink node* en el caso de que falle alguna de las operaciones efectuadas en la base de datos o que los datos contenidos en la petición GET no sean los esperados.

De esta forma, la funcionalidad de “config.php” se utiliza en el mecanismo de configuración (que se explica en la siguiente sección) y, posteriormente, el *sink node* envía los mensajes de los nodos cliente al servidor HTTP justo después de haberlos recibido y justo antes de volver a activar el modo *sleep*. Estos mensajes quedan registrados en la base de datos gracias al archivo “dataServer.php”.

3.4.3. Mecanismo de configuración de la WSN

En último lugar, el desarrollo de la solución propuesta se centró en implementar en la WSN el código necesario para la descarga de los parámetros de configuración y su envío desde el *sink node* hasta los nodos cliente.

El mecanismo de configuración se encarga de que los parámetros básicos almacenados en la tabla “config” de la base de datos lleguen a sus respectivos nodos cliente y estos realicen los ajustes oportunos para que la WSN quede completamente configurada.

Cuando los Waspmites se encienden a la vez al poner en marcha el sistema propuesto, el *sink node* se conecta a la red Wi-Fi y realiza la petición GET al servidor HTTP para obtener los parámetros de configuración, justo después de haber configurado la hora en el RTC. Mientras los nodos cliente esperan (durante un minuto), el *sink node* recibe los datos, los almacena y realiza los ajustes necesarios para más adelante conocer cuándo debe conectarse con cada uno de los nodos y así establecer correctamente las alarmas. Es entonces cuando se vuelve a conectar a la red Wi-Fi y establece un servidor TCP durante 45 segundos (si se supera ese tiempo la sesión TCP se reiniciará). Los nodos cliente terminan su espera e intentan unirse como clientes TCP a la conexión y enviar un mensaje indicando su nombre. Sin embargo, el servidor solo procesará el primer mensaje que reciba y responderá a ese nodo con un mensaje que contiene su minuto de conexión. Al recibir este dato, el cliente cierra la sesión TCP, configura su alarma y entra en modo *sleep* hasta que se produzca la interrupción del RTC. El *sink node* reinicia su conexión TCP (también se desconecta y se vuelve a conectar a la red) y realiza el mismo proceso tantas veces como nodos queden por configurar en la red. Mientras se van configurando todos los nodos, los que no son capaces aún de establecer conexión con el *sink node* esperan 10 segundos a una respuesta, después concluyen su sesión TCP y esperan 8 segundos hasta volver a realizar otro intento. Pasado un determinado período de tiempo, que será mayor cuanto mayor sea el número de nodos de la WSN, toda la red de sensores quedará configurada y sincronizada.

Es importante mencionar que se puede producir el siguiente caso especial: que exista una comunicación entre sensores programada para un minuto concreto pero el mecanismo de configuración aún no haya finalizado. Si se da este caso se pueden producir dos situaciones: si el nodo cliente aún no ha recibido su minuto de conexión simplemente el primer envío de datos al *sink node* se producirá en la siguiente hora (cuando la WSN esté configurada); la segunda situación sería aquella en la que el nodo cliente ya ha recibido su minuto de conexión pero el *sink node* aún no ha terminado de enviar los parámetros de configuración al resto de nodos, en la cual el nodo cliente interpreta que la conexión ha sido errónea y guarda la medición para enviársela al *sink node* junto con la siguiente. Además, el nodo servidor está configurado para establecer las conexiones con el resto de nodos según el minuto de conexión de cada uno en orden ascendente. Esto quiere decir que, si el mecanismo de configuración finaliza después del primer minuto de conexión, el *sink node* no se comunicará con el resto de nodos hasta una hora después, cuando sí pueda detectar la interrupción de esa primera comunicación.

Finalmente, con el mecanismo que se acaba de explicar, el desarrollo de la solución concluye. Para una mejor comprensión se puede consultar en el anexo III el diagrama de flujo del programa implementado. El código C++ elaborado puede ser cargado en los Waspmites con ayuda del IDE (asegurándose de que los ajustes iniciales en cada nodo, de los que se hablará en

la siguiente sección, son correctos) y, al encender todos los nodos de forma simultánea, el sistema completo comenzará a funcionar a pleno rendimiento.

3.5. Análisis del código implementado en los Waspmites

En este punto se va a analizar el código C++ (que se puede consultar en el anexo III) implementado en los Waspmites, ya que se trata de la parte del proyecto que más tiempo ha necesitado para su desarrollo y la que cumple el objetivo principal indicado en el primer capítulo de este documento. Primero se enumerarán los valores que es necesario especificar justo antes de cargar el código en el Waspmite correspondiente, después se indicarán las bibliotecas importadas y la función que cumple cada una de las constantes y variables globales y, por último, se explicarán brevemente las principales funciones que componen el código.

3.5.1. Ajustes iniciales

Para que el funcionamiento de los sensores inalámbricos sea el correcto, el usuario debe de introducir unos valores imprescindibles para la ejecución del programa justo antes de cargar el código en cada Waspmite. Estos valores se encuentran justo después de las bibliotecas incluidas, agrupados y acompañados por un comentario que indica “Ajustes”, tal y como se puede apreciar en la siguiente captura del código:

```
//-----> AJUSTES <-----  
// Para ambos modos  
char name[] = "COM6"; //Nombre del nodo  
#define SERVER_MODE true //Servidor(true)/cliente(false)  
// Para modo Servidor:  
#define MAX_NUM_CLIENTS 5 //Núm. máx de nodos cliente  
// Para modo Cliente:  
#define CLIENT_IP "192.168.1.52" //IP nodo cliente (predet.)  
//-----
```

Ilustración 15. Captura de los ajustes del código C++.

Se trata de cuatro valores, aunque solo será necesario modificar tres de ellos. El primero, un *array* de tipo *char*, es el nombre del nodo, que tiene que tener una longitud de 4 caracteres exactamente (se ha establecido así). El segundo, de tipo *boolean*, establece el modo de funcionamiento del nodo, es decir, deberá ser *true* si actúa como *sink node* (modo servidor) o *false* si se trata de un cliente. Si se ha seleccionado el modo servidor, se deberá indicar una última constante numérica: el número máximo de nodos cliente que podrá soportar la red. Es importante recalcar que este número no indica el número de clientes que formarán la WSN, ya que esa información se la proporciona el servidor HTTP al *sink node*. Por último, si el nodo va a formar parte de la red como cliente, se debe especificar su dirección IP ya que esta será fija.

Es necesario justificar por qué se deben especificar estos valores y no se descargan directamente del servidor HTTP como ocurre con los minutos de conexión. Por un lado, se hace imprescindible que exista un valor fijo en cada nodo que sirva como identificador para que el servidor pueda identificar qué fila de la tabla “config” de la base de datos contiene su

información. El valor elegido para desempeñar esta función es el nombre del nodo “name”, que también es utilizado por el *sink node* para identificar las peticiones de los nodos cliente. Por otro lado, el fichero de código que se carga en los Wasmotes sirve para los dos tipos de sensores, pero es necesario indicar en la constante “SERVER_MODE” qué modo es el elegido para cada dispositivo. La constante “MAX_NUM_CLIENTS” simplemente controla que el número de nodos cliente de la tabla “config” de la base de datos no sea superior a este valor. Por razones que se explicarán en el capítulo 4, se estima que la WSN diseñada puede llegar a funcionar con hasta 30 nodos cliente conectados al *sink node*. En último lugar, debido a que era necesario que los sensores conociesen la IP del *sink node* para poder establecer una conexión TCP con él, se decidió que todas las direcciones IP de los nodos fuesen fijas, al igual que la del nodo servidor. Por esta razón se debe fijar al valor de la constante “CLIENT_IP”.

3.5.2. Bibliotecas importadas y constantes y variables globales

Las bibliotecas que ha sido necesario incluir en el código son las siguientes:

WaspWiFi.h	Es necesaria para el manejo del módulo Wi-Fi.
WaspRTC.h	Permite manejar y configurar el RTC incorporado en el Waspote.
string.h	Es imprescindible para la utilización y el manejo de variables tipo <i>string</i> .
stdio.h	Es necesaria para utilizar algunas funciones para el manejo de variables <i>string</i> .

Tabla 6. Bibliotecas importadas.

Por otro lado, las constantes globales utilizadas (además de las explicadas en la sección anterior “Ajustes iniciales”) son las siguientes:

ESSID	Nombre de la red Wi-Fi.
AUTHKEY	Contraseña de la red Wi-Fi.
NETMASK	Máscara IP de la red Wi-Fi.
GATEWAY	Puerta de enlace (dirección IP del <i>router</i>) de la red Wi-Fi.
SERVER_IP	Dirección IP del nodo servidor.
SERVER_PORT	Puerto del nodo servidor para la conexión TCP.
CLIENT_PORT	Puerto del nodo cliente para la conexión TCP.
SERVER_SHORT_TIMEOUT	Tiempo de espera corto del servidor TCP (45 segundos).
SERVER_TIMEOUT	Tiempo de espera del servidor TCP (90 segundos).
CLIENT_SHORT_TIMEOUT	Tiempo de espera corto del cliente TCP (10 segundos).
CLIENT_TIMEOUT	Tiempo de espera del cliente TCP (30 segundos).

Tabla 7. Constantes globales.

En último lugar, antes de la primera función que se ejecutará al iniciar el Waspote, se encuentran declaradas las siguientes variables globales (tabla 8 en la siguiente página):

socket	<i>uint8_t</i>	Indica el <i>socket</i> del módulo Wi-Fi utilizado.
status	<i>uint8_t</i>	Se utiliza para comprobar la ejecución de algunos métodos.
previous	<i>unsigned long</i>	Se utiliza para guardar referencias de tiempo.
HOST	<i>char</i>	Contiene la URL del <i>host</i> (modo servidor ¹³).
urlConfig	<i>char</i>	Indica la dirección web del archivo “config.php” dentro del <i>host</i> (modo servidor).
urlData	<i>char</i>	Indica la dirección web del archivo “dataServer.php” dentro del <i>host</i> (modo servidor).
clientName	<i>char</i>	Se utiliza para guardar temporalmente el nombre del cliente TCP (modo servidor).
clientsName	<i>char</i>	Es un <i>array</i> de dos dimensiones que permite almacenar los nombres de los nodos cliente que componen la red ordenados según su minuto de conexión (modo servidor).
myAlarm	<i>char</i>	Se utiliza para guardar el minuto de conexión del nodo cliente y puntualmente en el modo servidor.
alarms	<i>char</i>	Se trata de un <i>array</i> bidimensional que contiene los minutos de conexión ordenados de los nodos cliente (modo servidor).
receiver	<i>char</i>	Se utiliza como <i>buffer</i> para la recepción de mensajes.
message	<i>char</i>	En él se almacena el último mensaje recibido (modo servidor) o que se va a enviar (modo cliente).
oldMessage	<i>char</i>	Contiene un mensaje que no pudo ser transmitido en la anterior conexión TCP.
body	<i>char</i>	Se utiliza como <i>buffer</i> para el envío de mensajes.
config	<i>char</i>	Contiene el mensaje de configuración elaborado por el servidor HTTP (modo servidor).
numAlarms	<i>int</i>	Número de minutos de conexión, es decir, número de nodos cliente existentes en la red (modo servidor).
configuredAlarm	<i>int</i>	Se utiliza temporalmente para identificar la alarma a configurar (modo servidor).
numMessages	<i>int</i>	Número de mensajes recibidos/enviados.
messageReceived	<i>boolean</i>	Se utiliza para chequear si se ha recibido un mensaje.

Tabla 8. Variables globales.

3.5.3. Descripción breve de las funciones implementadas

En este punto se proporcionará una descripción breve de las funciones que componen el código C++ implementado en los Waspmites. Generalmente, estos dispositivos están preparados para ejecutar dos funciones: *setup()*, que se ejecuta solo una vez cuando se enciende el Waspmite, y *loop()*, que se ejecuta justo después y cuando termina vuelve a ejecutarse creando un bucle infinito [42]. Sin embargo, para una mejor comprensión del programa y por practicidad, el código implementado consta de otras funciones. A continuación, se irá describiendo brevemente la funcionalidad de cada una en el orden en el que están escritas en el código:

¹³ Al mencionarse un modo entre paréntesis se quiere hacer notar que dicha variable únicamente se utiliza en ese modo de funcionamiento.

1. Función *setup()*:

Primero realiza la configuración de la hora en el RTC (encendiendo antes su módulo). Si se trata del *sink node*, ejecuta la función *downloadConfig()*, mediante la cual recibe el mensaje de configuración del servidor HTTP, y luego establece un servidor TCP (que se reinicia cada 45 segundos) tantas veces como sea necesario para recibir una a una las peticiones de configuración¹⁴ del resto de nodos hasta que la red esté completamente configurada. Por otro lado, si se trata de un nodo cliente, se realiza una espera de 60 segundos para que el *sink node* reciba los parámetros de configuración y luego se intenta establecer una conexión TCP con él. En cada intento se espera un mensaje de respuesta durante 10 segundos y luego se cierra la sesión TCP y se realiza otra espera de 8 segundos. Finalmente, para ambos nodos, se apaga el módulo Wi-Fi.

2. Función *loop()*:

Es la función más importante del código. En ella el *sink node* configura la siguiente alarma del *array* “alarms” (o la primera si ya lo ha recorrido entero), mientras que en el modo cliente se vuelve a configurar la alarma asignada a ese nodo. El Wasp mote entra en modo *sleep* (se apagan todos los módulos) y ‘despierta’ con la interrupción del RTC. Se borran los *flags* de interrupción y se encienden los módulos del RTC, de la conexión USB y, en último lugar, de comunicación (Wi-Fi). Es entonces cuando el nodo servidor ejecuta la función *server()* y, si recibe el paquete de datos del cliente, lo envía a la base de datos mediante la función *sendMessageToServer()*. Por otra parte, el nodo cliente espera 15 segundos al *sink node* y ejecuta la función *client()* (que se explica más adelante).

3. Función *downloadConfig()*:

Esta función solo se ejecutará una vez y únicamente lo hará el *sink node*. En ella el nodo servidor se conecta a la red Wi-Fi para enviar una petición GET al servidor HTTP indicando su nombre como parámetro. Se recibirá la respuesta del *host* (que contiene los nombres de los nodos cliente y su minuto de conexión ordenados por este último parámetro) y se guardarán los nombres en el *array* “clientsName” y los minutos de conexión en el *array* “alarms”. También se guardará el número de alarmas (o lo que es lo mismo, de nodos cliente) en la variable “numAlarms”. Por último, el *sink node* se desconecta de la red Wi-Fi.

4. Función *sendConfig()*:

Esta parte del código C++ también será únicamente ejecutada por el *sink node*. En ella se establecerá el servidor TCP y se esperará durante 45 segundos a que llegue alguna petición de configuración. Si esto sucede, se buscará en el mensaje el nombre del nodo cliente y se comparará con cada uno de los almacenados en el *array* “clientsName”. Cuando se encuentre dicho nombre en el *array*, se obtendrá el valor del *array* “alarms” que se encuentre en esa misma posición (su minuto de conexión) y se enviará al cliente TCP. Finalmente, se cerrará la sesión TCP, como hubiese ocurrido si después de los 45 segundos no se hubiese recibido ninguna petición.

¹⁴ Cuando el *sink node* recibe una petición de configuración de alguno de los Waspmotes éste se desconecta y se vuelve a conectar a la red Wi-Fi. Este proceso evita errores al establecer las siguientes conexiones TCP.

5. Función *receiveConfig()*:

Al contrario que las dos últimas funciones explicadas, ésta solo será ejecutada por los nodos cliente. En ella primero se establece el cliente TCP y se envía la petición de configuración al *sink node* (que contiene como parámetro el nombre del nodo). Justo después, se realiza una espera de 10 segundos para recibir una respuesta. Si esto ocurre, se guarda el minuto de conexión en la variable “myAlarm” y se da por concluida la conexión con el servidor TCP; si no se recibe respuesta alguna, se cierra directamente la sesión TCP.

6. Función *server()*:

Nuevamente esta función solo es utilizada por el *sink node*. En primer lugar, el nodo se conecta a la red Wi-Fi y establece un servidor TCP que permanecerá a la espera de recibir datos durante 90 segundos. Si esto no ocurre, se cerrará la sesión TCP y se abandonará la red; no obstante, si recibe un paquete de datos, antes de desconectarse comprobará el nodo origen de dichos datos (y guardará su nombre en la variable “clientName”) y cuántos mensajes (mediciones) contiene. Guardará el mensaje recuperado (si existiese) en la variable “oldMessage” y el mensaje más reciente en “message”. También almacenará el número de mensajes recibidos en “numMessages”.

7. Función *client()*:

Se trata de una función utilizada únicamente por los nodos cliente y cuyo primer paso es elaborar el mensaje que se va a enviar al *sink node* (en previsión de que pudiese ocurrir algún error con el módulo Wi-Fi). Inmediatamente después, el nodo se conecta a la red e intenta unirse a la sesión TCP del *sink node* como cliente. Si lo consigue, comprobará el valor de “numMessages” y enviará el paquete de datos con el número de mensajes correspondiente. Esperará 30 segundos a la confirmación de recepción del *sink node*. Si la recibe se desconectará de red, y si no guardará el mensaje no enviado en la variable “oldMessage” e introducirá el valor “2” en “numMessages” antes de desconectarse.

8. Función *sendMessageToServer()*:

Esta función configurará el módulo Wi-Fi para enviar las mediciones al servidor HTTP. Después se conectará a la red y comprobará el valor de “numMessages”: si es “2” se añadirán los caracteres “MENSAJE_RECUPERADO:” al mensaje guardado en “oldMessage”, se adecuará¹⁵ la cadena de caracteres y se enviará el mensaje al host mediante el método GET (de HTTP); justo después de esto se procederá de igual forma con el mensaje más reciente (sin añadir caracteres) como si “numMessages” tuviese valor “1”. Por último, se volverá a configurar el módulo Wi-Fi para establecer una conexión TCP en modo servidor. Como se puede comprobar por su funcionalidad, esta función solo es ejecutada por el *sink node*.

9. Función *notifyByLEDS()*:

Esta sencilla función permite informar al usuario del transcurso de la ejecución por medio de dos LED del Wasp mote. Si su primer parámetro (de tipo *boolean*) tiene valor “true”, un LED rojo parpadeará cinco veces para notificar un error. Si por el contrario es “false”, un LED verde parpadeará tantas veces como indique el segundo argumento de la

¹⁵ Se sustituirán los espacios en blanco contenidos en el mensaje por un guión bajo “_” para que los datos sean leídos correctamente por el archivo “dataServer.php” en el servidor HTTP.

función (de tipo *integer*). Con la intención de estandarizar la señalización que proporcionan los LED, se suelen notificar algunas operaciones de la misma forma:

- Si el LED verde se enciende 3 veces de forma consecutiva significa que el nodo se ha conectado a la red Wi-Fi.
- Si el LED verde se enciende 4 veces de forma consecutiva significa que se ha unido a una sesión TCP.
- Si el LED verde se enciende 6 veces de forma consecutiva significa que se ha recibido un mensaje en el *buffer* de recepción.
- Por otro lado, como ya se ha mencionado, si parpadea el LED de color rojo se está notificando que se ha producido algún tipo de error.

Después de la explicación de las 9 funciones que componen el código implementado en los Waspmites, se recomienda consultar el diagrama de flujo y el código completo del programa que se proporcionan en el anexo III de este documento para una comprensión más completa del sistema.

Por último, también es importante recordar que si se ejecuta el código mientras el Waspmite está conectado al ordenador es posible consultar el *monitor serial* del IDE para obtener información de la ejecución por pantalla. De cualquier manera, gracias a la última función del código, se puede comprobar fácilmente si se produce algún tipo de error durante el transcurso del programa. Es imprescindible asegurarse de que toda la red queda correctamente configurada ya que, si esto no se cumple, el *sink node* podría no concluir el proceso de configuración inicial y nunca comenzar a recibir las mediciones periódicas.

3.6. Resumen del capítulo

En este capítulo se ha descrito el diseño de comunicación sincronizado que se propone en este trabajo con el objetivo de ahorrar energía en las baterías de los nodos de la red de sensores. Una vez fijados los requisitos generales se ha procedido a explicar el prototipo implementado basado en el diseño propuesto y se han enumerado las diferentes etapas de desarrollo que han sido necesarias para su desarrollo. Por último, se ha explicado el funcionamiento del programa, cuyo código se puede consultar en el anexo III.

4. PRUEBAS DE FUNCIONAMIENTO

Una vez finalizado el proceso de diseño y desarrollo de la solución implementada en este trabajo, el siguiente paso que marca el objetivo principal del proyecto es el de comprobar el correcto funcionamiento del sistema. Los detalles de dichas comprobaciones se explican en este capítulo.

4.1. Pruebas realizadas

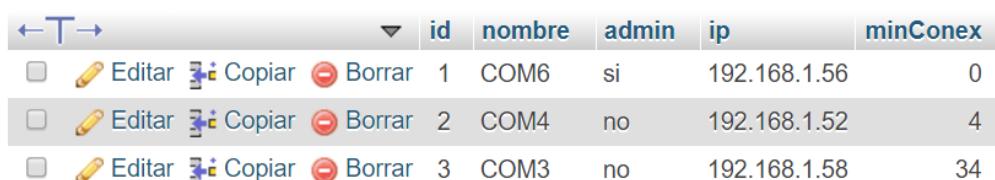
Según se fue avanzando en el desarrollo de la solución propuesta se fueron probando los archivos de código que se iban escribiendo. Primero se comprobó el intercambio de información en la red de sensores, luego se chequearon las comunicaciones del *sink node* con el servidor web y, por último, el mecanismo de configuración de la WSN explicado en el capítulo anterior. Sin embargo, los resultados relevantes para este trabajo se obtuvieron en las pruebas realizadas al sistema completo.

4.1.1. Ajustes previos

El primer paso que se llevó a cabo fue preparar el sistema implementado mediante una serie de ajustes previos al comienzo de las pruebas para que los resultados de las mismas pudieran ser comparados entre sí.

Para la sincronización del reloj RTC en todos los Waspmotes se fijó la siguiente fecha inicial: martes 4 de julio de 2017 a las 12:00 horas. Teniendo en cuenta que la ejecución del mecanismo de configuración de la WSN dura como mínimo cerca de dos minutos (aunque puede durar muchos más minutos dependiendo de la cantidad de nodos clientes de la red), se procuró que la primera conexión entre nodos se produjese cuatro minutos después del inicio del programa. Este ajuste se realizó por comodidad a la hora de poder comprobar rápidamente (evitando esperar una hora a que el *sink node* pudiese detectar la primera interrupción¹⁶) que la medición de esta primera conexión quedaba registrada en la base de datos.

En la siguiente imagen se muestra una captura de pantalla del contenido de la tabla "config" de la base de datos del servidor web en una de las pruebas efectuadas:



				id	nombre	admin	ip	minConex
<input type="checkbox"/>	Editar	Copiar	Borrar	1	COM6	si	192.168.1.56	0
<input type="checkbox"/>	Editar	Copiar	Borrar	2	COM4	no	192.168.1.52	4
<input type="checkbox"/>	Editar	Copiar	Borrar	3	COM3	no	192.168.1.58	34

Ilustración 16. Tabla "config" durante una prueba de funcionamiento.

¹⁶ Situación explicada en la [sección 3.4.3](#).

Sirva como ejemplo de los parámetros de configuración los datos de los nodos que se muestran en la tabla de la imagen superior. En las pruebas se ha utilizado como nombre de los nodos el de su interfaz puerto serie que utilizan al conectarse al ordenador (que tiene una longitud de 4 caracteres). También se especifican las direcciones IP correspondientes debido a que también se pasaba como parámetro a los nodos en algunas pruebas realizadas durante el desarrollo de la WSN, pero en el prototipo implementado dichos valores no se llegan a utilizar ya que las direcciones IP vienen pre-configuradas en el código interno de cada Wasp mote. El *sink node* no tiene asignado un minuto de conexión ya que utiliza los del resto de nodos, por eso su valor es "0" y en ningún caso el sistema llega a consultar dicho campo.

Existe un detalle importante que está relacionado con los minutos de conexión que también se muestran en la ilustración 16. Se estima que una conexión periódica entre el *sink node* y otro sensor puede llegar a durar como máximo algo más de 90 segundos (ya que este es el *timeout* del nodo servidor), lo que implica que los minutos de conexión tienen que distar al menos en dos unidades entre ellos, es decir, si una conexión periódica está configurada para el minuto 35 no podrá realizarse la siguiente hasta el minuto 37, y así sucesivamente. Esto fija el número máximo de nodos cliente que podría tener la WSN con esta configuración en 30 nodos¹⁷.

Como contenido de las mediciones de los sensores (ya que no se acopló ningún otro módulo más a los Wasp motes) se decidió incluir el valor del RTC y el porcentaje de batería disponible, ambos valores referentes al momento de elaboración del mensaje en cada nodo cliente. De esta manera, la 'medición' que se guarda en la base de datos tiene la siguiente forma (donde en color azul se distingue la información recopilada del RTC y de la batería):

Datos recopilados a *Tue, 04/07/17, 16: 02: 27* con un *38%* de batería

Junto a todos los ajustes que se acaban de mencionar también se incluyeron los especificados en el apartado 3.5.1. (los ajustes previos a cargar el código en cada Wasp mote). Para ello se asignó un nombre y una dirección IP distintos a cada nodo y que no estuviesen siendo utilizados por ningún otro dispositivo conectado a la red de infraestructura proporcionada por el *router*.

Después de tener en cuenta todos estos aspectos, el sistema quedó correctamente preparado para la realización de las pruebas. Este proceso puede repetirlo fácilmente cualquier persona si desea crear un entorno de pruebas similar al que se expone en este trabajo, siempre que disponga del hardware necesario y de los archivos de código que se proporcionan en el anexo III de la memoria.

Finalmente, todas las pruebas efectuadas tuvieron una puesta en marcha idéntica: se encendieron todos los nodos a la vez para que estuviesen correctamente sincronizados.

4.1.2. Registro de pruebas

Con el objetivo de comprobar el correcto funcionamiento del sistema implementado se estipularon los tres siguientes tipos de pruebas según su duración:

¹⁷ 60 minutos / 2 minutos por conexión = 30 conexiones posibles.

- ✓ Pruebas de 3 días de duración.
- ✓ Pruebas de 4 días de duración.
- ✓ Pruebas de una semana de duración.

Se llevaron a cabo varias pruebas de 3 y 4 días de duración y 2 pruebas de una semana de duración para comprobar la fiabilidad del sistema.

También se realizó otra prueba de 3 días de duración con el fin de comprobar si, tal y como se ha planteado en este proyecto, el ahorro de energía conseguido en el sistema implementado, respecto a otras soluciones asíncronas que mantienen el módulo de comunicación encendido todo el tiempo, era mucho mayor. Esta prueba se explica en detalle en la sección 4.2.1.

4.2. Resultados obtenidos

Los resultados obtenidos durante las pruebas de funcionamiento de la solución implementada son satisfactorios. Las pruebas pudieron ser finalizadas correctamente. Se produjo algún fallo aislado en el mecanismo inicial de configuración, pero reiniciar el sistema fue suficiente para remediarlo.

A continuación, se analizan los resultados obtenidos según la etapa del programa evaluada:

- En primer lugar, el mecanismo de configuración inicial de la red funciona correctamente, pero ha presentado un fallo al intentar configurar el sistema. Dicho fallo ocurre cuando el *sink node* recibe una petición de configuración, pero es otro nodo cliente diferente el que recibe el minuto de conexión mientras también estaba intentando obtener el suyo¹⁸. En este caso el *sink node* considera que un nodo más ha sido configurado y habrá dos nodos clientes que tendrán configurado el mismo minuto de conexión. Esto ha ocurrido únicamente en dos ocasiones. No obstante, el problema podría aparecer con mayor frecuencia en un sistema que constase de un número mayor de nodos cliente (hay que recordar que el prototipo implementado solo consta de dos nodos de este tipo).

Una peculiaridad del mecanismo es que los dos clientes TCP intentan unirse a la sesión iniciada por el servidor TCP, por lo que usualmente se produce algún tipo de colisión entre ellos que bloquea la conexión. Sin embargo, esto ya estaba contemplado en el diseño del mecanismo y por eso en los sucesivos intentos de conexión que se realizan los nodos se termina estableciendo las conexiones sucesivamente. No obstante, hay que considerar que este problema alarga el tiempo utilizado por el mecanismo de configuración y, en el caso de que haya un número mayor de nodos cliente que intenten unirse a la sesión TCP de forma simultánea el retraso causado podría ser mayor.

Por otra parte, nunca se han producido fallos en el proceso de descarga de parámetros de configuración por parte del *sink node*.

¹⁸ Esta situación ha ocurrido cuando los dos nodos cliente han enviado su petición de configuración prácticamente en el mismo instante y la sesión TCP ha sufrido algún tipo de desajuste.

- En segundo lugar, una vez la red de sensores se ha terminado de configurar no se han registrado fallos. Una medición se perderá únicamente cuando se sucedan dos intentos de conexión con el *sink node* fallidos, ya que el sistema solo es capaz de almacenar una medición para su posterior recuperación en el siguiente envío de datos al nodo servidor. Solo un fallo permanente en el módulo de comunicación de un nodo cliente inhabilita dicho nodo, pero un fallo de este tipo en el *sink node* dejaría inoperativa a toda la red de sensores.

- En último lugar, no se ha producido ningún tipo de fallo en el envío de mediciones al servidor web. La posibilidad de consultar en tiempo real la información recopilada por los sensores en la base de datos permite al usuario comprobar que el sistema funciona de forma correcta, es decir, ha servido como herramienta para monitorizar el funcionamiento de la solución implementada además de como un registro de datos.

En el anexo IV se proporciona, a modo de ejemplo, el contenido de la tabla “data” de la base de datos correspondiente a la prueba de tres días de funcionamiento ininterrumpido del prototipo implementado.

4.2.1. Prueba de comprobación del ahorro de energía

Para justificar el aumento de autonomía conseguido en los Waspotes mediante el uso del modo *sleep* para el ahorro de energía, se ha llevado a cabo una comprobación muy sencilla. Tal y como se ha mencionado en la sección 4.1.2, se realizó una prueba de funcionamiento del sistema, con una duración de tres días, donde se modificaron algunas líneas del código cargado en los sensores inalámbricos para evitar el uso del modo *sleep* y, en su lugar, mantener los nodos en modo activo con su módulo de comunicación encendido durante la espera hasta la siguiente conexión programada (imitando el funcionamiento de un sistema asíncrono).

Los cambios realizados en el código de los Waspotes se muestran a continuación en una sección del mismo (concretamente en la función *loop()*), donde se señalan tachadas y en color rojo las líneas eliminadas y con color verde las líneas añadidas:

```
// RUTINA PRINCIPAL DE EJECUCIÓN
void loop() {

    // Modo Servidor
    if (SERVER_MODE && numAlarms>0) {

        // Configuración de la alarma (asignadas de forma consecutiva)
char settingAlarm1[11];
strcpy(settingAlarm1, "00:00:");
strcat(settingAlarm1, alarms[configuredAlarm]);
strcat(settingAlarm1, ":00");
RTC.setAlarm1(settingAlarm1, RTC_ABSOLUTE, RTC_ALM1_MODE4);
USB.printf("\nAlarm1: %s\n", RTC.getAlarm1());
        alarmChecked = false;
        while (!alarmChecked) {
            strcpy(checkMinute, RTC.getTime());
            for (int i = 0; i<30; i++) {
                if (checkMinute[i]==':') {
                    if (checkMinute[i+1]==alarms[configuredAlarm][0] &&
                        checkMinute[i+2]==alarms[configuredAlarm][1]) {
```

```

        USB.println(F("ALARM!!"));
        notifyByLEDS(false,2);
        alarmChecked = true;
        delay(60000);
    }
    i = i+5;
}
}
}
configuredAlarm++;
if(configuredAlarm>=numAlarms)
    configuredAlarm = 0;
}

// Modo Cliente
else if (!SERVER_MODE) {

    // Configuración de la alarma
char settingAlarm1[11];
strcpy(settingAlarm1,"00:00:");
strcat(settingAlarm1,myAlarm);
strcat(settingAlarm1,":00");
RTC.setAlarm1(settingAlarm1,RTC_ABSOLUTE,RTC_ALM1_MODE4);
USB.printf("\nAlarm1: %s\n", RTC.getAlarm1());
    alarmChecked = false;
    while (!alarmChecked) {
        strcpy(checkMinute, RTC.getTime());
        for (int i = 0; i<30; i++) {
            if (checkMinute[i]==':') {
                if (checkMinute[i+1]==myAlarm[0] &&
checkMinute[i+2]==myAlarm[1]) {
                    USB.println(F("ALARM!!"));
                    notifyByLEDS(false,2);
                    alarmChecked = true;
                    delay(60000);
                }
                i = i+5;
            }
        }
    }

}

else {
    USB.println(F("No function expected"));
}

// Waspote en Modo Sleep
USB.println(F("Waspote goes to sleep...\n"));
notifyByLEDS(false,2);
PWR.sleep(ALL_OFF);

// ----->SLEEP MODE<-----

// Configuración de módulos tras Modo Sleep
RTC.ON();
USB.ON();
USB.println(F("Waspote wakes up!!"));

// Configuración post-interrupción
if( intFlag & RTC_INT ) {

```

```

intFlag &= ~(RTC_INT); // Limpiar flag
USB.println(F("-----"));
USB.println(F("RTC INT Captured"));
USB.println(F("-----"));

// Módulo WiFi ON
if( WIFI.ON(socket) == 1 ) {

USB.println(F("\nWiFi switched ON\n"));

// Modo Servidor
if (SERVER_MODE && numAlarms>0) {

```

En la siguiente tabla se muestran las mediciones registradas en la base de datos durante las primeras doce¹⁹ horas de operatividad del sistema sin hacer uso del modo *sleep*:

id	nombre	datos	hora
1	COM4	Datos recopilados a Tue, 04/07/17, 12:05:17 con un 65% de batería	7:07:31
2	COM3	Datos recopilados a Tue, 04/07/17, 12:35:17 con un 65% de batería	7:37:31
3	COM4	Datos recopilados a Tue, 04/07/17, 13:05:17 con un 64% de batería	8:07:30
4	COM3	Datos recopilados a Tue, 04/07/17, 13:35:17 con un 64% de batería	8:37:30
5	COM4	Datos recopilados a Tue, 04/07/17, 14:05:17 con un 63% de batería	9:07:30
6	COM3	Datos recopilados a Tue, 04/07/17, 14:35:17 con un 63% de batería	9:37:30
7	COM4	Datos recopilados a Tue, 04/07/17, 15:05:17 con un 63% de batería	10:07:30
8	COM3	Datos recopilados a Tue, 04/07/17, 15:35:17 con un 62% de batería	10:37:31
9	COM4	Datos recopilados a Tue, 04/07/17, 16:05:17 con un 62% de batería	11:07:30
10	COM3	Datos recopilados a Tue, 04/07/17, 16:35:17 con un 61% de batería	11:37:30
11	COM4	Datos recopilados a Tue, 04/07/17, 17:05:17 con un 60% de batería	12:07:30
12	COM3	Datos recopilados a Tue, 04/07/17, 17:35:17 con un 60% de batería	12:37:31
13	COM4	Datos recopilados a Tue, 04/07/17, 18:05:17 con un 59% de batería	13:07:30
14	COM3	Datos recopilados a Tue, 04/07/17, 18:35:17 con un 59% de batería	13:37:30
15	COM4	Datos recopilados a Tue, 04/07/17, 19:05:17 con un 58% de batería	14:07:33
16	COM3	Datos recopilados a Tue, 04/07/17, 19:35:17 con un 58% de batería	14:37:30
17	COM4	Datos recopilados a Tue, 04/07/17, 20:05:17 con un 58% de batería	15:07:33
18	COM3	Datos recopilados a Tue, 04/07/17, 20:35:17 con un 58% de batería	15:37:30
19	COM4	Datos recopilados a Tue, 04/07/17, 21:05:17 con un 56% de batería	16:07:33
20	COM3	Datos recopilados a Tue, 04/07/17, 21:35:17 con un 57% de batería	16:37:30
21	COM4	Datos recopilados a Tue, 04/07/17, 22:05:17 con un 55% de batería	17:07:30
22	COM3	Datos recopilados a Tue, 04/07/17, 22:35:17 con un 55% de batería	17:37:32
23	COM4	Datos recopilados a Tue, 04/07/17, 23:05:17 con un 54% de batería	18:07:30
24	COM3	Datos recopilados a Tue, 04/07/17, 23:35:17 con un 54% de batería	18:37:30

Tabla 9. Registro de mediciones (sin modo *sleep*).

¹⁹ Se muestran únicamente 12 horas a modo de muestra, por comodidad, pero la prueba tuvo una duración de 3 días. Además, se ha obviado la columna “fecha” para que la tabla no ocupase varias páginas.

Los resultados obtenidos son muy diferentes en lo que se refiere al gasto de energía en las baterías. Mientras que utilizando el modo *sleep* la carga de la batería de los nodos desciende de media un 1% cada 24-28 horas (como se puede comprobar en el registro adjuntado en el anexo IV a modo de ejemplo), sin utilizar este modo la batería pierde de media ese mismo porcentaje de carga en menos de una hora.

A la vista de estos datos, queda demostrado que el sistema síncrono diseñado en este trabajo mejora las prestaciones energéticas de otros sistemas similares que sí mantienen el módulo de comunicación encendido continuamente para poder recibir datos.

4.3. Conclusiones del capítulo

En este capítulo se ha descrito el entorno de pruebas en el que se ha comprobado el funcionamiento del prototipo descrito en el capítulo 3: ajustes previos, contenido de los mensajes, duración de las pruebas, etc. A continuación, se han expuesto los resultados obtenidos. El mecanismo de configuración ha registrado un fallo aislado pero el sistema funciona correctamente y los datos recopilados son satisfactorios. En último lugar, se ha descrito la prueba realizada sin el modo *sleep* con el fin de comparar el prototipo implementado con un sistema similar que no apaga su módulo de comunicación. El resultado también ha sido muy positivo ya que se ha obtenido un ahorro de energía muy importante, justificando el diseño del sistema.

5. CONCLUSIONES

En este capítulo se interpretan los resultados obtenidos en las pruebas enumeradas en el capítulo anterior y se analiza el resultado final del proyecto extrayendo las conclusiones finales y proponiendo algunas mejoras que podrían perfeccionar el sistema en un futuro.

5.1. Conclusión principal

La conclusión principal que se puede extraer después del diseño, desarrollo y prueba del sistema implementado es que éste cumple el objetivo principal y casi todos los objetivos más concretos que fueron estipulados en el comienzo del proyecto.

La solución de comunicación sincronizada propuesta es capaz de transportar los datos recopilados por los nodos cliente a través de todo el esquema de conexiones del sistema hasta su almacenamiento en la base de datos del servidor web, minimizando el coste energético respecto a otros diseños (tal y como se ha demostrado en la sección 4.2.1.).

5.2. Conclusiones secundarias

En esta sección se analizarán otras conclusiones relacionadas con el resto de objetivos del trabajo y otras observaciones de interés.

El único objetivo de los planteados al inicio del proyecto que no se ha podido cumplir es el que hace referencia a las conexiones tipo *ad-hoc*. No se ha conseguido implementar en los nodos cliente la comunicación directa entre ellos sin hacer uso del *router* Wi-Fi y, por tanto, de la red de infraestructura. De esta forma, todos los nodos deben estar dentro del rango de cobertura del *router* para poder formar parte de la WSN (no pudiéndose recopilar información en zonas que estén fuera de este perímetro) y no se realizan conexiones con un esquema que no sea el de cliente-servidor. Aunque esto sería una limitación en un sistema real, en este caso particular no afecta a la validación del sistema, pues se ha mantenido la comunicación Waspote-Waspote aunque fuera a través del punto de acceso.

El resto de objetivos del trabajo se han cumplido sin problemas. Uno de ellos es el que especifica que las mediciones deben ser almacenadas en una base de datos y se deben poder consultar en cualquier momento y en tiempo real. Otro se ha cumplido gracias a la posibilidad que ofrece el sistema de configurar el minuto de conexión de cada nodo cliente directamente desde la base de datos, sin necesitar de modificar el código cargado en los Waspmotes. No obstante, sí que es necesario que dichos minutos de conexión sean fijados antes de poner en marcha el sistema, ya que este parámetro se envía a cada nodo durante el mecanismo de configuración que se ejecuta solo una vez al iniciarse el sistema.

Por último, el diseño implementado se ha podido probar en numerosas ocasiones con el fin de comprobar su correcto funcionamiento, pero también se han valorado otras características a raíz de los resultados obtenidos:

- Robustez:

El sistema de comunicación sincronizado muestra una gran precisión a la hora de realizar todas las conexiones necesarias para mantener el flujo de información que se establece entre los nodos cliente y el servidor web (extremos del esquema de conexiones). Prueba de ello es que las mediciones se registran en la base de datos prácticamente sin retraso²⁰ alguno. Sin embargo, como se ha mencionado en el capítulo anterior, la solución propuesta está preparada para afrontar errores de conexión durante la etapa de realización de las mediciones pero no tanto durante el mecanismo de configuración, donde es más vulnerable. Es recomendable que el usuario se asegure de que el sistema queda correctamente configurado observando las señalizaciones proporcionadas por los LED o comprobando más tarde si las mediciones se están registrando en la base de datos.

- Tiempo de operatividad:

Según los resultados de las pruebas realizadas, la batería de los Waspnotes consume de media un 1% de su carga cada 24-28 horas. Por medio de un cálculo²¹ matemático sencillo se puede concluir que el tiempo de operatividad máximo del sistema sin cargar las baterías es algo superior a 100 días naturales. No obstante, las pruebas se han realizado con baterías recargables de 6600mA/h pero se podrían utilizar otras con distinta capacidad que harían variar el tiempo de operatividad del sistema. Además, dichas baterías también se deterioran con el paso del tiempo y en algunos casos no pueden volver a cargarse completamente (se quedan en cifras cercanas al 100% de carga).

- Rango de cobertura:

Como ya se ha mencionado recientemente, el rango de cobertura de la WSN coincide con el del *router* Wi-Fi ya que los nodos se conectan entre ellos haciendo uso de la red de infraestructura proporcionada por dicho dispositivo. El perímetro de dicha red Wi-Fi dependerá de la marca y el modelo del *router* utilizado y de los obstáculos físicos que existan en el entorno de la red.

- Escalabilidad:

Una vez puesta en marcha la red no se pueden añadir más nodos a ella. Es necesario reiniciar el sistema, al igual que en el caso de los parámetros de configuración. Por otra parte, la red de sensores puede llegar a soportar teóricamente hasta 30 nodos cliente, pero, a la vista de los resultados obtenidos, se puede concluir que el mecanismo de configuración podría colapsarse o no finalizar correctamente con un número de nodos cliente superior al utilizado en las pruebas (quizás no con 3 o 4 pero sí con más).

²⁰ Esto se puede comprobar en la columna "hora" del registro de mediciones adjuntado en el anexo IV.

²¹ 1 día de operatividad por cada 1% de consumo da lugar a 100 días de operatividad.

5.3. Posibles mejoras

En este último apartado del capítulo se proponen algunos cambios o mejoras que se podrían desarrollar en un futuro con el fin de optimizar el sistema ya implementado. Además, al final de la sección se propone un posible trabajo a realizar que conllevaría desarrollar un diseño más avanzado que el propuesto en esta memoria. Para organizar las ideas que se van a exponer se ha decidido clasificarlas según la etapa del programa en la que se aplicarían.

En primer lugar, se van a mencionar algunas posibles mejoras aplicables al mecanismo de configuración del diseño implementado:

- Una de las principales mejoras que se podría aplicar al protocolo de configuración sería que éste fuese capaz de poder actualizar los parámetros de todos los nodos cada cierto tiempo. Por ejemplo, una forma de implementarlo consistiría en que el *sink node* descargase los parámetros de configuración periódicamente y, de esta forma, cuando recibiese una medición, la respuesta enviada al nodo cliente podría contener la información de configuración actualizada correspondiente a ese nodo. De esta manera el sistema no tiene que ser reiniciado para que los cambios en la configuración de la WSN puedan ser llevados a cabo.

- Otra de las principales mejoras consistiría simplemente en aumentar el número de parámetros de configuración que se podrían modificar en la base de datos. Por ejemplo, si los Waspmotes tuviesen más módulos acoplados se podría especificar desde la base de datos cuáles de ellos se deben utilizar y, por lo tanto, qué datos se quieren recibir. Esto permitiría que el sistema diseñado pudiese utilizarse para más de una aplicación simultáneamente (por ejemplo, monitorizar la temperatura de una estancia y a la vez registrar cuándo hay movimiento en la misma).

- Una función que podría ayudar a mejorar la fluidez del mecanismo de configuración se fundamentaría en que los nodos cliente comprobasen si son capaces de conectarse a la red Wi-Fi. En caso de que esto no fuese posible continuarían ejecutando el mecanismo de configuración establecido, pero en el caso de que sí pudiesen conectarse a internet a través del *router* el nodo podría conectarse directamente al servidor web y recibir el mensaje de configuración de la red (tal y como lo hace el *sink node*). Así el sensor podría buscar en el mensaje sus parámetros (mediante su nombre) y facilitar el envío de peticiones al *sink node* por parte de sus compañeros. Sin embargo, todo este proceso se debería llevar a cabo antes de que el *sink node* descargase todos los parámetros y, además, se le tendría que notificar de alguna forma en ese mismo mensaje (se podría añadir una columna a la tabla “config” que indicase si el nodo ya está configurado o no).

- Otra mejora que solucionaría la mayoría de errores a la hora de establecer la sesión TCP entre nodo cliente y *sink node* consistiría en fijar una ventana de conexión para la configuración de cada nodo cliente. Esto quiere decir que cada uno de ellos mandaría su petición de configuración al *sink node* en un momento determinado en el que el resto de nodos cliente no estuviesen intentando mandar también su petición. Para llevar a cabo esta mejora solo sería necesario modificar la parte del mecanismo de configuración que afecta a los nodos cliente ideando algún método para la asignación

de esperas aleatorias que, estadísticamente y dependiendo del número de nodos de la red, consiguiese evitar que dos o más nodos realizasen peticiones al *sink node* simultáneamente.

En segundo lugar, las posibles mejoras que se mencionan a continuación corresponden a las aplicables durante la etapa de conexión periódica de los nodos cliente con el *sink node*:

➤ Una de las mejoras a implementar más importantes tiene relación directa con el mecanismo de sincronización del sistema. En las pruebas realizadas en este proyecto la solución diseñada se ha mantenido correctamente sincronizada en todo momento. No obstante, la utilización de este sistema en una aplicación real a largo plazo podría originar un desfase entre los módulos RTC de cada nodo que podría afectar a la sincronización de la WSN. Sería muy positivo el desarrollo de un mecanismo²² mediante el cual los nodos fuesen capaces de eliminar dicho desfase teniendo como referencia la hora configurada en el *sink node*. Además, esta información de sincronización podría incluirse en la respuesta que envía el *sink node* al nodo cliente en las conexiones periódicas.

➤ Otra mejora importante que se podría implementar en la red de sensores consistiría en programar conexiones *ad-hoc* entre los Waspnotes. Esto permitiría a los nodos poder realizar mediciones fuera del rango de cobertura del *router* Wi-Fi y conectarse a la red a través de otro nodo cercano a él. Con esta configuración de conexiones la WSN abandonaría la topología en estrella y podría adoptar otra tipo árbol o tipo malla. Sin embargo, al realizar este cambio organizativo en la red el sistema implementado ya no sería tan sencillo y ganaría en complejidad.

➤ Una pequeña mejora que se podría ejecutar de forma sencilla se fundamentaría en cambiar el modo *sleep* por el modo hibernación en el proceso de ahorro de energía. Aunque son muy similares, este último consigue reducir el consumo a $0,07\mu\text{A}$ en comparación con el modo *sleep* ($55\mu\text{A}$) [12] y su implementación también es muy parecida.

➤ Otra opción que mejoraría el rango de cobertura de la red consistiría en la instalación de repetidores Wi-Fi. Por medio de ellos la red de infraestructura ampliaría su perímetro de cobertura y ningún ajuste extra sería necesario en la WSN.

➤ Por último, otra mejora que se podría estudiar se basaría en que los nodos cliente se pudiesen desconectar temporalmente del sistema cuando su carga estuviese próxima a agotarse y así realizar un cambio de batería. El principal problema que esto conllevaría sería el de posibilitar su vuelta al sistema, pero con una comprobación periódica del *sink node* podría llegarse a implementar. A su vez, también se podría estudiar el caso en el que el *sink node*, que acumula un mayor gasto de batería, pudiera desconectarse temporalmente para que su batería fuese reemplazada o realizar una petición para un paso de testigo y que otro nodo le sustituyese en el papel de *sink node*.

²² Similar al explicado en la sección 2.4. de este documento.

En tercer y último lugar, se van a mencionar dos posibles mejoras que se podrían aplicar en la etapa de registro de mediciones en la base de datos en la que intervienen el *sink node* y el servidor web:

➤ La primera mejora que se podría llevar a cabo en esta etapa del programa se basaría en la creación de otras tablas similares a la tabla “data” en las que se guardarían mediciones de diferente tipo. Por ejemplo, si un Wasp mote tiene acoplado un módulo para la medición de la presión atmosférica y otro capaz de medir la humedad podría registrar las mediciones de ambos en tablas separadas para facilitar la consulta de los datos. Esto se podría conseguir fácilmente modificando el archivo “dataServer.php”.

➤ La segunda estaría enfocada a proteger el acceso a la base de datos y también se podría llevar a cabo en la primera etapa del programa (mecanismo de configuración). Dicha mejora se fundamentaría básicamente en solicitar al *sink node* una contraseña para poder registrar mediciones en la base de datos y así proteger la información de posibles accesos malintencionados.

5.3.1. Proposición de un sistema más avanzado

Un futuro trabajo interesante con relación al elaborado en este Proyecto de Fin de Grado consistiría en generalizar el sistema de comunicaciones propuesto. Se eliminaría completamente el concepto de *sink node* y solo existirían dos tipos de nodos: unos con la capacidad de comunicarse con la red de infraestructura y otros que solo podrían comunicarse con otros nodos. Cada uno de ellos tendría un período en el que escucharía peticiones que tendría que notificar a sus vecinos y, si procede, al servidor accesible a través de la infraestructura. Para enviar un mensaje a un nodo se elegiría un instante aleatorio dentro de su período de escucha (con el fin de reducir las posibles colisiones). Todos los nodos podrían enviar información (conociendo cuál es el período de recepción del nodo destinatario) o recibir (en su propio período de recepción). Los datos podrían ser: una medición a enviar o reenviar al servidor, una solicitud de datos de configuración, o un envío de datos de configuración (directamente desde el servidor o a través de otros nodos).

6. BIBLIOGRAFÍA

- [1] L. M. Gracia, «¿Qué es Waspote?», UN POCO DE JAVA Y +, 21 Agosto 2012. [En línea]. Available: <https://unpocodejava.wordpress.com/2012/08/21/que-es-waspote/>. [Último acceso: 2 Agosto 2017].
- [2] National Instruments, «¿Qué es una Red de Sensores Inalámbricos?», National Instruments, 22 Abril 2009. [En línea]. Available: <http://www.ni.com/white-paper/7142/es/>. [Último acceso: 3 Agosto 2017].
- [3] C. Buratti, A. Conti, D. Dardari y R. Verdone, «An Overview on Wireless Sensor Networks Technology and Evolution», National Center for Biotechnology Information, 31 Agosto 2009. [En línea]. Available: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3290495/>. [Último acceso: 3 Agosto 2017].
- [4] Y. Touati, A. Ali-Chérify B. Daachi, Energy Management in Wireless Sensor Networks, ISTE Press Ltd., 2017.
- [5] Libelium, «Waspote», Libelium, [En línea]. Available: <http://www.libelium.com/products/waspote/>. [Último acceso: 4 Agosto 2017].
- [6] G. F. Coulouris, Sistemas distribuidos: conceptos y diseño, Addison Wesley, 2001.
- [7] Área Tecnología, «¿Qué es WiMax?», Área Tecnología, [En línea]. Available: <http://www.areatecnologia.com/informatica/wimax.html>. [Último acceso: 4 Agosto 2017].
- [8] CCM - Comunidad informática, «Redes inalámbricas», CCM - Comunidad informática, 27 Julio 2017. [En línea]. Available: <http://es.ccm.net/contents/818-redes-inalambricas#tipos-de-redes-inalambricas>. [Último acceso: 4 Agosto 2017].
- [9] Masadelante.com, «¿Qué es el ancho de banda?», Masadelante.com, [En línea]. Available: <http://www.masadelante.com/faqs/ancho-de-banda>. [Último acceso: 4 Agosto 2017].
- [10] ComputerHoy, «¿Qué es 802.11ac y qué lo hace tan rápido?», ComputerHoy, 1 Febrero 2014. [En línea]. Available: <http://computerhoy.com/noticias/internet/que-es-wifi-80211ac-que-hace-tan-rapido-8789>. [Último acceso: 20 Septiembre 2017].
- [11] A. Crespo, «Latencia: Por qué existe y cómo funcionan los modos fastpath e interleaving», RedesZone, 7 Octubre 2011. [En línea]. Available: <https://www.redeszone.net/2011/10/07/latencia-por-que-existe-y-como-funcionan-los-modos-fastpath-e-interlaving/>. [Último acceso: 4 Agosto 2017].
- [12] Libelium, «Waspote Datasheet (v12)», Julio 2016. [En línea]. Available: http://www.libelium.com/downloads/documentation/v12/waspote_datasheet.pdf. [Último acceso: 6 Agosto 2017].

- [13] R. Verdone, D. Dardari, G. Mazzini y A. Conti, *Wireless Sensor and Actuator Networks : Technologies, Analysis and Design*, Elsevier, 2008.
- [14] M. Roche, «Time Synchronization in Wireless Networks,» Washington University in St. Louis, 23 Abril 2006. [En línea]. Available: https://www.cse.wustl.edu/~jain/cse574-06/ftp/time_sync/index.html. [Último acceso: 7 Agosto 2017].
- [15] A. Rodríguez, «Cómo consultar el nivel de contaminación de Madrid,» ELMUNDO, 17 Noviembre 2015. [En línea]. Available: <http://www.elmundo.es/madrid/2015/11/17/564b5c6346163f46318b4606.html>. [Último acceso: 8 Agosto 2017].
- [16] A. Palazzesi, «Historia del GPS: Cómo el mundo dejó de perderse,» NEOTEO, 23 Julio 2010. [En línea]. Available: <http://www.neoteo.com/historia-del-gps-como-el-mundo-dejo-de-perderse/>. [Último acceso: 8 Agosto 2017].
- [17] N. Rivera, «Qué es el Internet of Things y cómo cambiará nuestra vida,» Hipertextual, 20 Junio 2015. [En línea]. Available: <https://hipertextual.com/2015/06/internet-of-things>. [Último acceso: 11 Agosto 2017].
- [18] BY, «¿Qué es RFID?,» BY, [En línea]. Available: <https://www.by.com.es/blog/que-es-rfid/>. [Último acceso: 11 Agosto 2017].
- [19] J. Fernández Bes, TESIS: Algoritmos Adaptativos Energéticamente Eficientes para Redes de Sensores Inalámbricas, Madrid: Universidad Carlos III de Madrid, 2015.
- [20] M. d. R. Arroyo Valles, TESIS: Modelos Estadísticos Selectivos Energéticamente Eficientes para las Comunicaciones en Redes de Sensores, Madrid: Universidad Carlos III de Madrid, 2011.
- [21] E. Romero Perales, TESIS: Estrategias Cognitivas para la Reducción del Consumo Energético en la Redes de Sensores Inalámbricas, Madrid: Universidad Politécnica de Madrid, 2015.
- [22] T. Rault, TESIS: Eficiencia Energética en las Redes de Sensores Inalámbricas, Compiègne: Universidad de Tecnología de Compiègne, 2015.
- [23] University of Southampton, «Glacsweb,» University of Southampton, [En línea]. Available: <http://glacsweb.org/>. [Último acceso: 16 Agosto 2017].
- [24] help4mood, «help4mood,» help4mood, [En línea]. Available: <http://help4mood.info/site/default.aspx>. [Último acceso: 16 Agosto 2017].
- [25] EL CONFIDENCIAL, «Crean un sistema para saber qué hacen los pacientes con depresión en su casa,» EL CONFIDENCIAL, 27 Enero 2014. [En línea]. Available: https://www.elconfidencial.com/sociedad/2014-01-27/crean-un-sistema-para-saber-que-hacen-los-pacientes-con-depresion-en-su-casa_80839/. [Último acceso: 16 Agosto 2017].

- [26] LYNCEUS, «LYNCEUS,» LYNCEUS, [En línea]. Available: <http://www.lynceus-project.eu/>. [Último acceso: 16 Agosto 2017].
- [27] M. Soeiro, «FUTURE-CITIES Report Summary,» European Comission: CORDIS, 11 Agosto 2016. [En línea]. Available: http://cordis.europa.eu/result/rcn/188104_en.html. [Último acceso: 16 Agosto 2017].
- [28] CCM - Comunidad informática, «Introducción a Wi-Fi (802.11 o WiFi),» CCM - Comunidad informática, 15 Octubre 2016. [En línea]. Available: <http://es.ccm.net/contents/789-introduccion-a-wi-fi-802-11-o-wifi>. [Último acceso: 17 Agosto 2017].
- [29] IEEE Standards Association, «IEEE-SA - The IEEE Standards Association,» IEEE Standards Association, [En línea]. Available: www.standards.ieee.org. [Último acceso: 18 Agosto 2017].
- [30] EcuRed, «Estándar inalámbrico 802.11b,» EcuRed, [En línea]. Available: https://www.ecured.cu/Est%C3%A1ndar_inal%C3%A1mbrico_802.11b. [Último acceso: 18 Agosto 2017].
- [31] IEEE Standards Association, «802.15.4-2011 - IEEE Standard for Local and metropolitan area networks--Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs),» IEEE Standards Association, 2011. [En línea]. Available: <http://standards.ieee.org/findstds/standard/802.15.4-2011.html>. [Último acceso: 18 Agosto 2017].
- [32] F. Martín Archundia Papacetzzi, «El estándar IEEE 802.15.4,» 16 Diciembre 2003. [En línea]. Available: http://catarina.udlap.mx/u_dl_a/tales/documentos/lem/archundia_p_fm/capitulo4.pdf. [Último acceso: 19 Agosto 2017].
- [33] C. P. García, «Zigbee, Comunicación para Dispositivos,» SG Buzz, [En línea]. Available: <https://sg.com.mx/content/view/310>. [Último acceso: 19 Agosto 2017].
- [34] International Electrotechnical Commission - White Paper, «Internet of Things: Wireless Sensor Networks,» [En línea]. Available: <http://www.iec.ch/whitepaper/pdf/iecWP-internetofthings-LR-en.pdf>. [Último acceso: 19 Agosto 2017].
- [35] IEEE, «802.15.1-2002 - IEEE Standard for Telecommunications and Information Exchange Between Systems - LAN/MAN - Specific Requirements - Part 15,» IEEE, 14 Junio 2001. [En línea]. Available: <http://ieeexplore.ieee.org/document/1016473/>. [Último acceso: 19 Agosto 2017].
- [36] Libelium, «Libelium Adds LoRaWAN for Full Compatibility with Smart Cities Networks,» Libelium, 24 Noviembre 2015. [En línea]. Available: <http://www.libelium.com/lorawan-waspmote-868-europe-900-915-us-433-mhz-asia-lora/>. [Último acceso: 19 Agosto 2017].
- [37] EcuRed, «IEEE - EcuRed,» EcuRed, [En línea]. Available: <https://www.ecured.cu/IEEE>. [Último acceso: 19 Agosto 2017].

- [38] Agencia Española de Protección de Datos, «Glosario de términos,» Agencia Española de Protección de Datos, [En línea]. Available: https://www.agpd.es/portalwebAGPD/canalresponsable/inscripcion_ficheros/preguntas_frecuentes/glosario/index-ides-idphp.php. [Último acceso: 19 Agosto 2017].
- [39] LOPD (Ley Oficial de Protección de Datos), «Qué es la LOPD,» LOPD (Ley Oficial de Protección de Datos), [En línea]. Available: <https://www.lopd-proteccion-datos.com/ley-proteccion-datos.php>. [Último acceso: 19 Agosto 2017].
- [40] G. Nunemacher, Introducción a las redes de área local, Paraninfo, 1997.
- [41] Libelium, «Waspote RTC Programming Guide (v12),» [En línea]. Available: http://www.libelium.com/downloads/documentation/v12/waspote-rtc-programming_guide.pdf. [Último acceso: Marzo 2017].
- [42] Libelium, «Waspote IDE User Guide (v12),» [En línea]. Available: http://www.libelium.com/downloads/documentation/v12/waspote_ide_user_guide.pdf. [Último acceso: Febrero 2017].
- [43] Computing, «El mercado de soluciones IoT moverá 7.100 millones de dólares en 2020,» Computing, 11 Julio 2017. [En línea]. Available: <http://www.computing.es/mercado-ti/informes/1099591046401/mercado-de-soluciones-iot-movera-7100-millones-de-dolares-2020.1.html>. [Último acceso: 14 Septiembre 2017].

ANEXO I: Resumen extendido en Inglés

The emergence of WLANs (Wireless Local Area Network) in the telecommunications sector marked a turning point in internet access. This event has caused, directly or indirectly, the development of new devices depending on the use of the network that is going to be made: smartphones, tablets, netbooks... But also other types of devices have been created that are not designed to be used directly by a user. These are small electronic devices that have the sole objective of collecting information and transmitting it to a processing unit. The digital interconnection of this type of devices and other everyday objects has been called IoT (Internet of Things).

It's estimated that IoT can cause a huge impact in the coming years, both economic and social, being able to move 7 billion dollars in 2020. The idea of "digital home" could easily become a reality thanks to IoT technology. Wireless communication allows the development of new applications that will replace some traditional jobs, such as a farmer's one. The cost of maintaining an electronic system composed of wireless devices to take the care of a crop is much lower than the salary of one or more workers. Because of this, the transcendence that IoT can have could be compared with the one that had the industrial revolution when it began in the 18th century.

This work focuses on a type of systems with much transcendence that are included in the concept of IoT: Wireless Sensor Networks (WSN). These networks are composed of autonomous distributed devices and each incorporates at least one sensor to monitor physical or environmental conditions. However, the wireless interconnection of these devices requires an amount of energy that could be a problem depending on the time needed to keep them on. The autonomy of these units is limited because they operate with batteries. For this reason, it's necessary to find a balance between benefits and energy saving to obtain an efficient monitoring system and to make it capable of being operational during the period of time required by the user.

One of the easiest ways to save battery on network nodes is to shut down their communications module while not sending or receiving data. However, it's often necessary for a device to be able to receive communications from other units. If a node's communication module is turned off when another node is trying to send some kind of information to it, a connection between them can never be established.

The solution to the mentioned problem that is contemplated in the design proposed in this work is based on configuring a synchronized communication scheme. This solution consists of scheduling a series of data-sharing periods in which the nodes will be able to transmit information. This saves energy during periods of time when there's no communication between the network nodes.

In addition, the work also demonstrates that the energy consumption resulting from applying the proposed system is much lower than that obtained with a similar design that does not turn off at any time its communication module.

For all these reasons, the proposed system takes the advantages of the modular opensource Waspmote platform and the Wi-Fi technology, decreasing the amount of energy needed in the nodes to maintain the operating network over time.

The structure of the work is organized into six chapters and four appendices whose contents are summarized below:

- **Chapter 1. Introduction.**

This first chapter contains a brief introduction to the main topic of the work. In addition, the purpose and the objectives are explained and, finally, the structure of the memory is described.

The purpose of this Final Degree Project is to develop and test a solution, as simple as possible, for WSNs. That solution must make nodes being able to transfer the measurements in a synchronized way and take them to a database. Besides that, it has to minimize the energy cost throughout all the process. This should be achieved by using the sleep mode available in Waspnotes during significant periods of time when no communications take place.

In the same way, the proposed solution will meet some more specific objectives. It must support ad-hoc communications, but it also needs to implement the necessary settings for nodes to use an infrastructure network. Likewise, network communications should be peer-to-peer. The system must also have access to a database, giving the possibility to store the information collected by the sensors. Another objective indicates that the sensors must be configurable (at least partially) from the server without having to modify the code implemented in the Waspnotes. Finally, it's specified that the system must be tested, after its design and elaboration, to demonstrate its validity and the advantage obtained in the autonomy of the batteries.

- **Chapter 2. State of art.**

This second chapter contains a basic study of the current situation of wireless sensor networks, related to the main objective of this project.

First, the concept of wireless sensor network has been explained in detail: the function of the nodes that compose it, the topologies used, its origin due to ad-hoc networks, etc.

These networks are characterized by their great flexibility and they offer a wide range of configuration possibilities. For example, the geographical position of the nodes could allow to charge their batteries using solar panels, although this requires them to incorporate some type of protection for adverse weather conditions. The sink node plays a very important role in the connection scheme, because it's a key piece in the network's data stream.

This type of systems are mainly used for two types of applications: event detection and spatial and temporal estimation of a process. They do not require high transmission speeds and they commonly require high energy efficiency and robustness.

The next section explains in more detail the Waspnote platform. These devices, developed by the spanish company Libelium, get one of the lowest consumptions of the IoT market. The nodes are composed of several modules: a processing unit, a rechargeable battery, at least one communication module and one or more sensor modules (from the 110 available provided by Libelium). In addition, to load the code in the devices is used an IDE almost identical to Arduino's one.

For the choice of the communication module, three factors are usually taken into account: the distance between nodes, because there are technologies such as BlueTooth that only achieve a coverage of 10 meters and others like 4G that can provide coverage at national level; the bandwidth, where, for example, the IEEE 802.11n standard (Wi-Fi) is capable of transmitting 600Mbps; and latency, which is the sum of temporary delays that a packet suffers when it's transmitted through the network.

Right after the description of the characteristics of the Wasp mote platform, the work explains the WSN's existing techniques for energy saving. There are 3 types:

- Those based on management and division of time, which use the sleep mode during periods when the node does not establish connections with its peers.
- Data-oriented techniques, which take into account the type of data recorded by the sensors. If the information is important, it will be transmitted and, otherwise, it will be ignored.
- Those that are based on the mobility of the sensors, which propose a set of optimal trajectories for the nodes that, in addition to distribute the consumption between the devices, also help to avoid collapses in the network due to the breakdown of one or more nodes.

The next point of the work analyzes the different options available to synchronize WSNs and maintain that synchronization. Several existing methods are analyzed, such as the Network Time Protocol (NTP) that periodically eliminates the discrepancy produced in the node's clocks, taking as a reference the hour configured in the sink node.

After this section, the work brings information about the different applications that wireless sensor networks currently have and classifies them in 7 scenarios:

- Environmental monitoring: They tend to develop outdoors, have large dimensions and usually their duration is several years. They have protection and security mechanisms and the information they obtain is accessible in real time. They detect or monitor some type of phenomenon or natural parameter that, with other methods, would require human supervision.
- Medical care: They are critical applications because human health depends on them. The system may not be stationary if it's used on patients outside of medical facilities. In addition, the nodes are usually very small because they can be placed near vital organs.
- Positioning and tracking of objects or living beings: the nodes of a WSN can be used as highly accurate locators. It can be deployed outdoors, for example to monitor animals, or indoors, where GPS (Global positioning System) does not get good results.
- Logistics: A product can be tracked from production phase to delivery phase with a reduced cost. The systems usually can deal with hundreds or thousands of products, using nodes of different types.

- Domestic environment: These systems allow remote control of a multitude of domestic aspects: power off or on lights, home appliances handling, heating control or electrical expenditure monitoring. This is one of the great advantages of IoT.
- Industrial and commercial environment: they allow, for example, the monitoring of machines in a factory and the generation of some kind of alarm when an anomalous data is recorded.
- Other fields of interest: such as entertainment, mood, transportation or office work.

The next section mentions some doctoral theses that are directly related to the main topic of this work and, just after, another section lists large projects (most of them current) that use sensors networks as a basis for the development of new systems.

The last part of the chapter, section 2.8. "Regulatory framework", contains a brief description of the most used technical standards in WSNs, and also an analysis of the legislation applicable to the proposed solution in this work.

Three standards are described: the IEEE 802.11 standard (Wi-Fi), the IEEE 802.15.4 standard (with ZigBee Technology) and the IEEE 802.15.1 Standard (BlueTooth). In addition, other existing standards and technologies are mentioned and also some versions of the three standards that have been described.

In the second part of the section it's informed that is not necessary to pay any license in order to implement the proposed solution, because IEEE 802.11 standard can be used freely and it makes use of a frequency range included in the ISM bands (Industrial, Scientific and Medical). It also mentions the aspects to be taken into account for data protection, with some useful safety and prevention measures.

- **Chapter 3. Design of the proposed solution.**

The first thing that is provided in this chapter is a clear description of the proposed system, which has already been provided at the beginning of this summary. The system is composed of client nodes, a sink node and a server with access to a database. Design's addressess are wireless sensors networks that have excessive energy consumption and need a simple solution that drastically reduces it.

The following section specifies the four general requirements that the system must satisfy in any of its applications: the first of them indicates that the nodes must support Wasmote-server and Wasmote-Wasmote connections; the second specifies the network configuration mechanism; the third fixes the guidelines for the storage and consultation of the information collected; finally, the fourth establishes that the network must be able to run up to 30 client nodes per sink node.

The rest of the chapter is an explanation of the test environment implemented.

First, a list containing the software and the hardware used is presented. The cost of these materials is detailed in appendix III of the project.

Just after, the work provides an explanation of the connection schema designed and the network data flow. The system consists of two client nodes, one sink node, a

gateway (Wi-Fi router) and a web server with access to a MySQL database (My Structured Query Language). The data originate in the sensors and reach the database through the sink node (which uses the gateway to access the internet) and the server.

In the next section (3.4.), the three main stages that compose the development of the test environment are described: the first, which details the building of the WSN and the exchange of periodic messages between client nodes and sink node (with its structure); the second, which describes the communication between sink node and web server (and the PHP code files that allow it); and the third, which explains the initial configuration mechanism of the network (this includes the download of the configuration stored in the database and its shipment from the sink node to the client nodes one by one, sequentially).

In the section 3.5, the main program code is analysed. The necessary parameters must be set in each Wasp mote. Then, imported libraries, constants and variables used in the code are listed, briefly describing the function of each one of them. Finally, a short description of the nine functions that compose the code C++ is provided.

- **Chapter 4. Functionality tests.**

This chapter details the checks made to the implemented prototype in the test environment.

The first part of the chapter describes the previous adjustments established, in order to obtain the maximum amount of information possible and a homogeneity in the tests (that allows the comparison of results). In addition, the three types of tests performed with different duration are listed.

The second part of the chapter explains the results obtained in the tests: there is an analysis about the three stages of the program, highlighting the strengths and explaining the errors detected. After that, a point describes the process of checking the energy saving achieved in the proposed design. This is a test that is identical to the rest, except that the sleep mode is not used and the communication module are not turned off during free communication periods. The results show that the use of sleep mode prologs the autonomy of the node batteries.

- **Chapter 5. Conclusions.**

This chapter is divided into two sections. The first one interprets the information obtained in the tests and analyzes the final result of the project, and the second one proposes some improvements for the system that could be applied in the future.

In the first section the main conclusion extracted from the implemented system is explained. The implementation fulfils the main objective and almost all the more specific objectives (that were stipulated at the beginning of the summary). The implemented design is capable of transporting the data collected by the client nodes, through the entire system connection scheme, to its storage in the web server database, and it also minimizes the energy cost in comparison to other designs.

In terms of the secondary conclusions, the node-node connection couldn't be implemented without using the infrastructure network, that is to say, it has not been able to establish ad-hoc connections. However, the rest of the objectives have been accomplished without problems. The system is robust and stays synchronized during the

tests performed. Besides, it's able to recover against errors, except one registered in the initial configuration mechanism. For this reason, is advisable for the user to ensure that initial configuration ends correctly. It's estimated that the maximum operation time (without charging the node batteries) of the system is approximately 100 days. Its coverage range is the same as the Wi-Fi router's one. Finally, once the system is started, no more nodes can be added to the network, which is capable of supporting a maximum of 30 client nodes.

In the second section of the chapter, the improvements are classified according to the stage of the program in which they could be applied. One of the most remarkable reforms would be to modify the configuration mechanism, so some parameters of the network could be modified at any time from the server. Another interesting improvement requires that client nodes check if they have access to the internet: if so, they download their configuration directly from the server. Other improvement, related to the synchronism of the system, would be based on eliminating the internal clock lag of the client nodes respect to the sink node periodically. Another recommended change would be to establish the ad-hoc connection that could not be implemented in the test environment. A proposal for improving security in the access to the database is also included.

Finally, a proposition for a new system is included at the end of the improvement section. It's an advanced design of the one has been implemented. In this new schema, the concept of sink node is eliminated and more communication options are available.

- **Chapter 6. Bibliography.**
- **Appendices.**

At the end of the memory, four attachments are included with the following supplementary content:

- Appendix I: Extended summary in English.
- Appendix II: Socio-economic environment:

It includes the project elaboration budget (fixed in 7.370,72€) and an analysis of the socio-economic impact that could have the application of the system proposed. The work developed does not have a direct economic objective, but the target is to investigate a mechanism of low consumption that can increase the flexibility of the IoT solutions.

- Appendix III: Implemented code.
- Appendix IV: Measurement record.

ANEXO II: Entorno socio-económico

En este anexo se incluye el desarrollo del presupuesto del proyecto y un análisis del impacto socio-económico esperado debido a la aplicación de dicho sistema.

No obstante, con el objetivo de aportar información sobre el trabajo realizado, en la siguiente tabla se adjunta el plan de tareas del proyecto, donde se detallan las horas dedicadas a cada etapa del trabajo realizado:

Tareas	Día de comienzo	Día de finalización	Horas dedicadas
Estudio inicial del entorno del proyecto	20/04/2017	25/04/2017	20
Diseño del sistema propuesto	26/04/2017	05/05/2017	40
Diseño del prototipo basado en la solución propuesta	06/05/2017	10/05/2017	20
Total → Diseño de la solución propuesta	20/04/2017	10/05/2017	80
Primeros pasos con Waspote	11/05/2017	20/05/2017	40
Elaboración del esquema de la WSN	21/05/2017	10/06/2017	84
Implementación de la gestión de los mensajes recibidos y enviados por los nodos	11/06/2017	15/06/2017	20
Configuración de la sincronización de los sensores inalámbricos	16/06/2017	20/06/2017	20
Implementación del mecanismo de recuperación de mensajes perdidos	21/06/2017	22/06/2017	8
Total → Elaboración de la WSN	11/05/2017	22/06/2017	172
Contratación y configuración del servidor HTTP	23/06/2017	25/06/2017	12
Elaboración de las tablas de la base de datos	26/06/2017	30/06/2017	20
Creación de los archivos PHP del servidor web	01/07/2017	10/07/2017	40
Establecimiento de la comunicación servidor – sink node	11/07/2017	15/07/2017	20
Total → Implementación del servidor HTTP	11/05/2017	15/07/2017	92
Elaboración de los mensajes de configuración	16/07/2017	20/07/2017	20
Creación del protocolo para la configuración secuencial de la WSN	21/07/2017	31/07/2017	44
Total → Implementación del mecanismo de configuración	11/05/2017	31/07/2017	64
Redacción de la memoria del proyecto	01/08/2017	15/09/2017	184
Revisión de la memoria del proyecto	16/09/2017	26/09/2017	44
Total → Elaboración de la memoria del proyecto	01/08/2017	26/09/2017	228
TOTAL PROYECTO	20/04/2017	26/09/2017	636

Tabla 10. Plan de tareas del proyecto.

Presupuesto del Trabajo de Fin de Grado

1.- Autor: Carlos Parra Marcelo

2.- Departamento: Telemática

3.- Descripción del Proyecto:

Comunicaciones Sincronizadas en
 - Título Dispositivos Limitados
 - Duración 5 meses
 - Tasa de costes indirectos: **20%**

4.- Presupuesto total del Proyecto (valores en Euros): **7.370,72 Euros**

5.- Desglose presupuestario (costes directos)

PERSONAL

Apellidos y nombre	N.I.F.	Categoría	Dedicación (personas/mes) a)	Coste persona mes	Coste (Euros)
Parra Marcelo, Carlos		Ingeniero	5	1.079,67	5.398,35
Soto Campos, Ignacio		Ingeniero Senior	0,5	1.379,67	689,84
Personas/mes			5,5	Total	6.088,19

a) 1 Persona mes = 131,25 horas. Máximo anual de dedicación de 12 personas mes (1575 horas)

Máximo anual para PDI de la Universidad Carlos III de Madrid de 8,8 personas mes (1.155 horas)

EQUIPOS

Descripción	Coste (Euro)	% Uso dedicado proyecto	Dedicación (meses)	Período de depreciación	Coste imputable d)
Kit Waspote 1	264,00	100	3	60	13,20
Kit Waspote 2	264,00	100	3	60	13,20
Kit Waspote 3	264,00	100	3	60	13,20
Total					39,60

d) Fórmula de cálculo de la Amortización:

$$\frac{A}{B} \times C \times D$$

A = nº de meses desde la fecha de facturación en que el equipo es utilizado

B = periodo de depreciación (60 meses)

C = coste del equipo (sin IVA)

D = % del uso que se dedica al proyecto (habitualmente 100%)

SUBCONTRATACIÓN DE TAREAS

Descripción	Empresa	Coste imputable
Total		0,00

OTROS COSTES DIRECTOS DEL PROYECTO^{e)}

Descripción	Empresa	Coste imputable
Subscripción de <i>hosting</i> (3 meses)	Hostinger	14,48
Total		14,48

^{e)} Este capítulo de gastos incluye todos los gastos no contemplados en los conceptos anteriores, por ejemplo: fungible, viajes y dietas, otros,...

6.- Resumen de costes

Presupuesto Costes Totales	Presupuesto Costes Totales
Personal	6.088,19
Amortización	39,60
Subcontratación de tareas	0
Costes de funcionamiento	14,48
Costes Indirectos	1.228,45
Total	7.370,72

Impacto socio-económico

Como se ha detallado en el capítulo 2, actualmente se están llevando a cabo multitud de proyectos para todo tipo de aplicaciones que hacen uso, de una forma u otra, de una red de sensores inalámbricos. Para ello se utilizan varias plataformas similares a Waspote, como por ejemplo Raspberry Pi o Arduino. El diseño que se propone en este proyecto es una solución básica sincronizada para la plataforma Waspote que pretende servir como base para el desarrollo de sistemas más complejos en los que sea necesario un notable ahorro de energía con el fin de aumentar la autonomía de los nodos, pero manteniendo la posibilidad (aunque sea a horas restringidas) de poder enviar o recibir información. El impacto socio-económico que puede causar esta solución de forma directa en su sector de aplicación puede ser bastante importante a largo plazo, ya que el desarrollo de otros sistemas más complejos y con un fin más específico que utilicen el trabajo que se ha llevado a cabo en este proyecto sí puede causar un gran impacto en la utilización de esta plataforma.

Respecto a su impacto económico, como se ha detallado en el presupuesto, el coste del sistema propuesto no es excesivamente elevado para la funcionalidad que puede llegar a proporcionar con su implementación (sobre todo en lo referente a gastos de material). Esta característica hace que dicho sistema sea atractivo a la hora de que un usuario evalúe si realizar una inversión en él, ya que fácilmente puede recuperar el dinero invertido con el paso del tiempo. Además, el sistema se puede integrar con alguna de las plataformas similares que se han mencionado en el párrafo anterior, pudiendo minimizar así el coste total del sistema (al aprovechar otros sensores que el usuario pudiese tener previamente) y aumentando la cobertura y la versatilidad de la WSN. Asimismo, el impacto económico que están teniendo las redes de sensores inalámbricos es enorme, ya que poco a poco se están reemplazando puestos de trabajo tradicionales que desempeñaban tareas que ahora son capaces de cumplir con un coste menor este tipo de redes. En este punto es indispensable hablar de IoT (*Internet of Things*), ya que se estima que en el año 2020 el mercado asociado a este concepto moverá 7.100 millones de dólares [43]. Un dato más que pone de manifiesto el inmenso impacto económico causado por los avances en este tipo de tecnologías.

Por otro lado, si se estudia el impacto ético y social que podría tener una aplicación real de este sistema también se debe hablar de IoT. El cambio que puede suponer la implantación de este concepto en la vida cotidiana de las personas ha sido y es objeto de estudio. Tanto es así que a menudo se equipara la transcendencia que puede llegar a tener este concepto con la que tuvo en su día la revolución industrial [17]. Sin embargo, surge un dilema ético cuando (como se ha mencionado en el párrafo anterior) un sistema similar al que se propone en este trabajo se utiliza para una aplicación que antes desempeñaba un trabajador. La función que era desempeñada por este empleado ahora se puede llevar a cabo de forma automatizada por una red de sensores y, con el paso del tiempo, dicho puesto de trabajo terminará por desaparecer. Este conflicto ético va ligado a muchos avances tecnológicos y forma parte del desarrollo del sistema laboral.

Por último, si se analiza el impacto medioambiental, las redes de sensores inalámbricos evitan la instalación de sistemas cableados que anteriormente afectaban al entorno natural de una u otra forma. Además, el objetivo principal del diseño implementado en este trabajo es ahorrar energía, lo que también minimiza la huella ecológica que pudiese causar la utilización de este sistema. Asimismo, en el capítulo 2 se han mencionado varias aplicaciones relacionadas

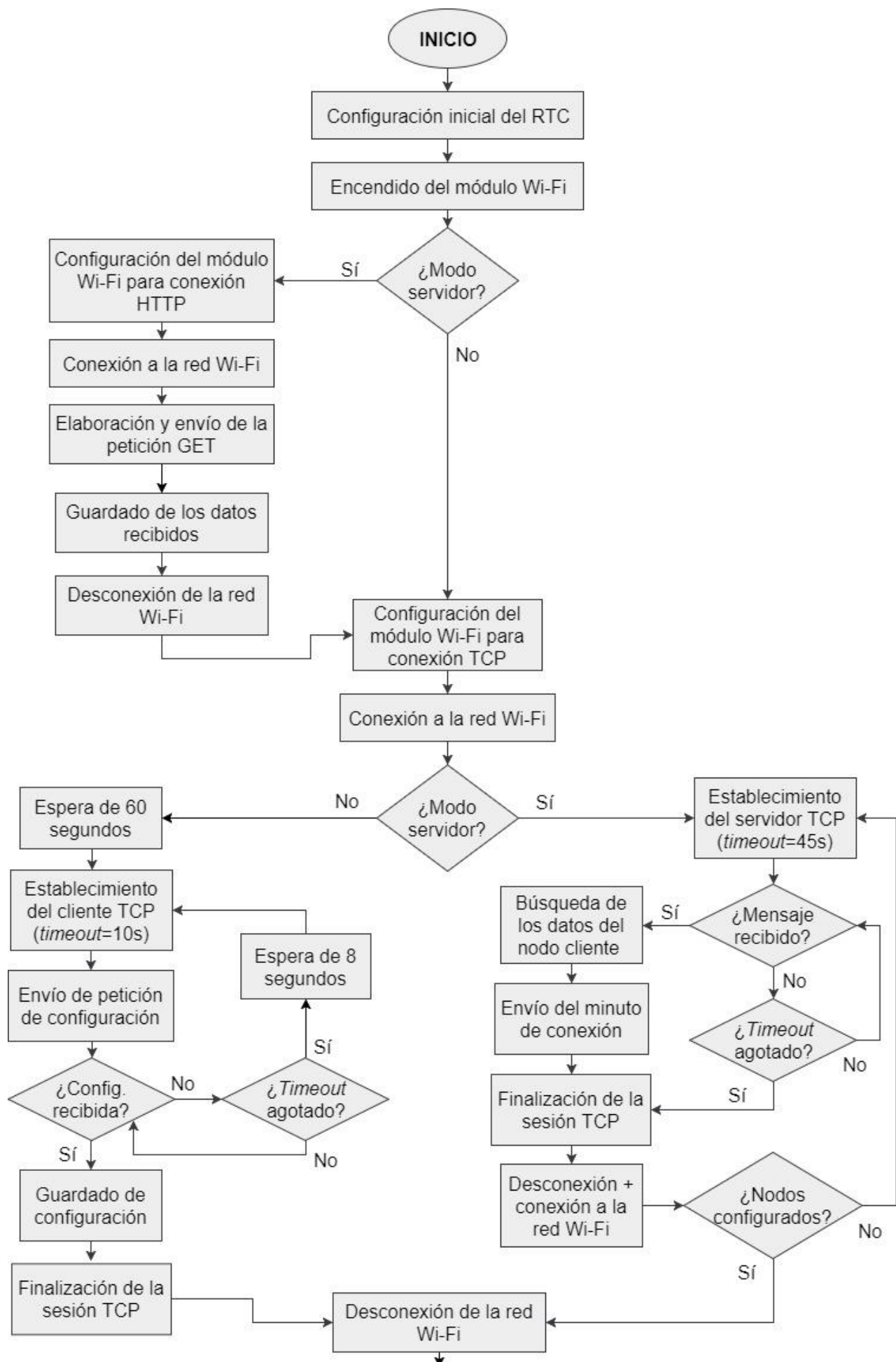
con el entorno medioambiental (como la monitorización de diversos aspectos de un ecosistema).

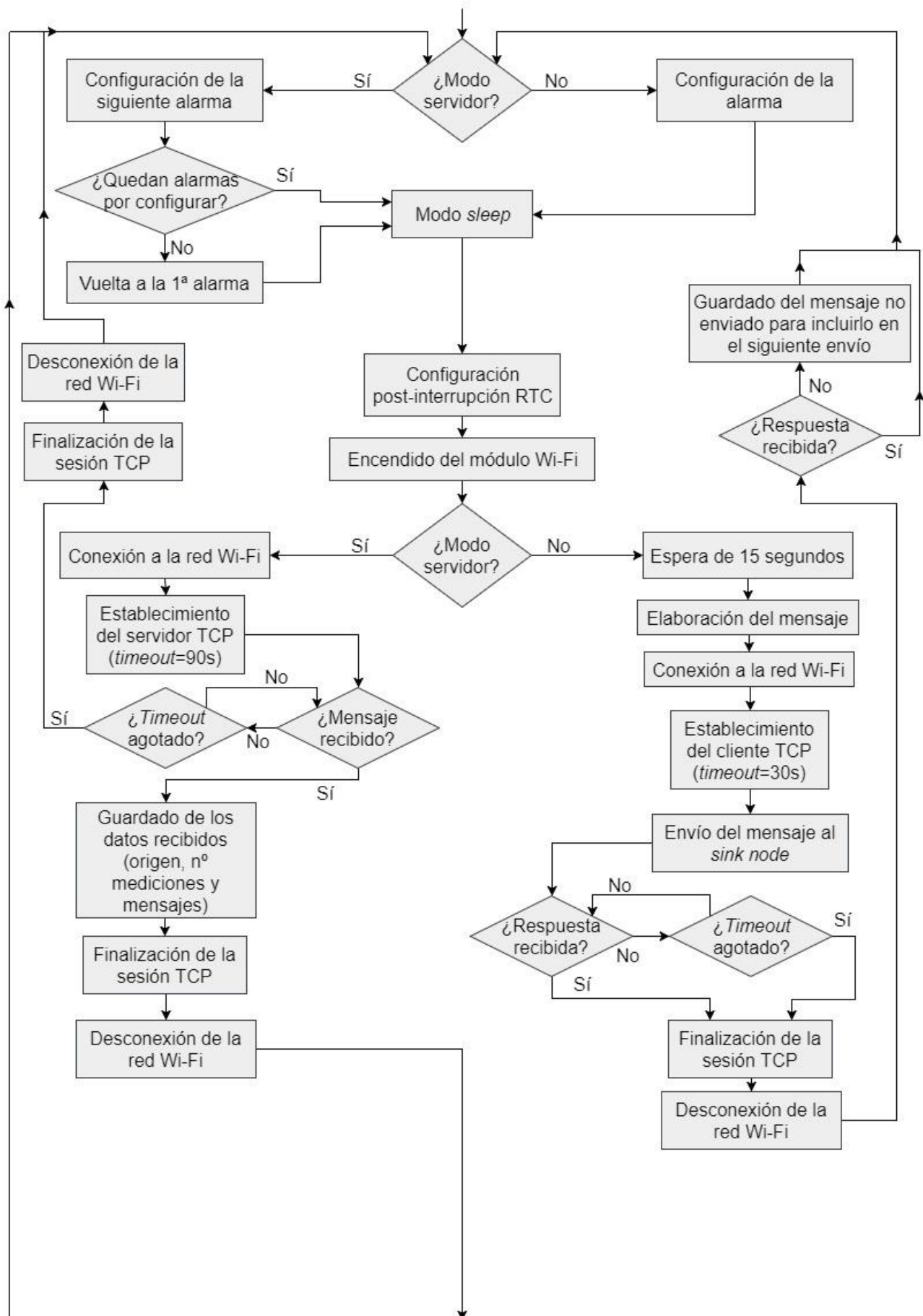
A modo de conclusión del apartado, es importante mencionar que el trabajo elaborado en este Proyecto de Fin de Grado no tiene un objetivo económico directo. El objetivo que se persigue con el desarrollo del trabajo es el de investigar un mecanismo de bajo consumo que pueda aumentar la flexibilidad de las soluciones IoT, lo que redundará finalmente en una ayuda al impacto que están causando estas tecnologías.

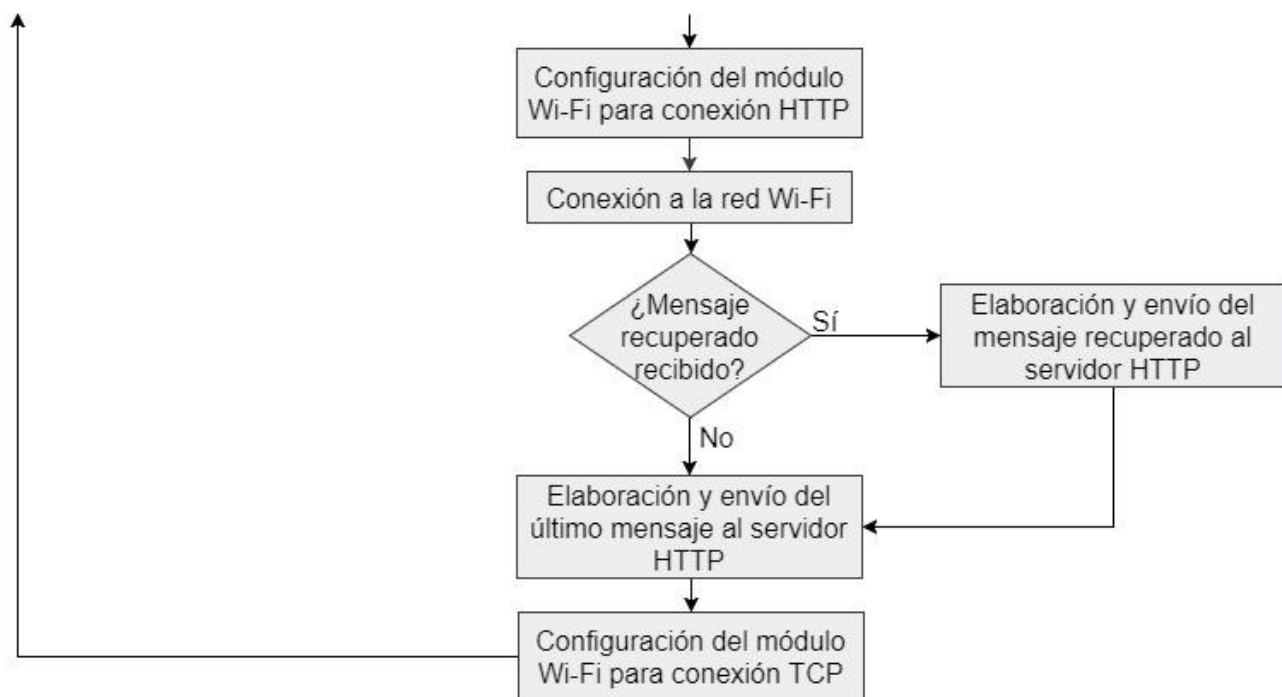
ANEXO III: Código implementado

Diagrama de flujo del programa principal

En las tres siguientes páginas de este anexo se representa el diagrama de flujo del código C++ cargado en los Waspmotes y que implementa el funcionamiento tanto de un *sink node* como de un nodo cliente, ya que solo se ha desarrollado un archivo de código para que pueda ser cargado en cualquiera de los nodos.







Código C++ del programa principal

```
// BIBLIOTECAS INCLUIDAS
#include <WaspWiFi.h>
#include <WaspRTC.h>
#include <string.h>
#include <stdio.h>

//-----> AJUSTES <-----
// Para ambos modos
char name[] = "COM4"; //Nombre del nodo
#define SERVER_MODE false //Servidor(true)/cliente(false)
// Para modo Servidor:
#define MAX_NUM_CLIENTS 5 //Núm. máx de nodos cliente
// Para modo Cliente:
#define CLIENT_IP "192.168.1.52" //IP nodo cliente (predet.)
//-----

// CONSTANTES Y VARIABLES GLOBALES
#define ESSID "XXXXXXXX" // Nombre de la red WiFi
#define AUTHKEY "XXXXXXXXXXXXXXXXXXXX" // Contraseña de la red WiFi
#define NETMASK "255.255.255.0" // Máscara de red
#define GATEWAY "192.168.1.1" // Puerta de enlace
#define SERVER_IP "192.168.1.56" //IP del nodo servidor
(predeterminada)
#define SERVER_PORT 2000 // Puerto del nodo servidor
#define CLIENT_PORT 3000 // Puerto del nodo cliente
#define SERVER_SHORT_TIMEOUT 45000 // Tiempo de espera corto
(servidor)
#define SERVER_TIMEOUT 90000 // Tiempo de espera (servidor)
#define CLIENT_SHORT_TIMEOUT 10000 // Tiempo de espera corto (cliente)
#define CLIENT_TIMEOUT 30000 // Tiempo de espera (cliente)

uint8_t socket = SOCKET0; // Socket del módulo WiFi
uint8_t status; // Variable de comprobación
unsigned long previous; // Variable de tiempo
char HOST[] = "carlinhopm.esy.es"; // Dirección del servidor HTTP
char urlConfig[] = "GET$/config.php?"; // Para descargar los datos de
configuración
char urlData[] = "GET$/dataServer.php?"; // Para subir mediciones a la
base de datos
char clientName[6]; // Nombre del nodo cliente actual (modo servidor)
char clientsName[MAX_NUM_CLIENTS][6]; // Registro de nombres de los
nodos cliente
char myAlarm[4]; // Alarma del nodo cliente
char alarms[MAX_NUM_CLIENTS][4]; // Registro de alarmas de los nodos
cliente
char receiver[513]; // Buffer de recepción
char message[507]; // Último mensaje enviado o recibido
char oldMessage[507]; // Penúltimo mensaje enviado o recibido
// (recuperación de mensaje perdido)
char body[513]; // Buffer de envío
char config[513]; // Mensaje de configuración (modo servidor)
int numAlarms; // Número de alarmas (número de nodos clientes de la
red)
int configuredAlarm; // Posición en el registro de la alarma
configurada
int numMessages; // Número de mediciones recibidas (servidor) o a
enviar (cliente)
```

```

boolean messageReceived; // Comprobación de mensaje recibido

// AJUSTES INICIALES
void setup() {

    USB.println(F("STARTING PROGRAM ---> Setup:"));

    // Configuración del RTC
    RTC.ON();
    RTC.setTime("04:07:17:03:12:00:00");
    USB.print(F("Time [Day of week, YY/MM/DD, hh:mm:ss]: "));
    USB.println(RTC.getTime());

    // Encendido del módulo WiFi
    if (WIFI.ON(socket)==1) {
        USB.println(F("\nWiFi switched ON\n"));

        // Modo Servidor (sink node)
        if (SERVER_MODE) {

            // Configuración del módulo WiFi (conexión HTTP)
            WIFI.resetValues();
            WIFI.setConnectionOptions(HTTP);
            WIFI.setDHCPoptions(DHCP_ON);
            WIFI.setJoinMode(MANUAL);
            WIFI.setAuthKey(WPA2, AUTHKEY);
            WIFI.storeData();

            // Descarga de los datos de configuración de la red y valores
            // iniciales
            downloadConfig();
            messageReceived = false;
            configuredAlarm = 0;

        }

        numMessages = 1; // Valor inicial

        // Configuración del módulo WiFi (conexión TCP)
        WIFI.resetValues();
        if (SERVER_MODE)
            WIFI.setConnectionOptions(CLIENT_SERVER);
        else
            WIFI.setConnectionOptions(CLIENT);
        WIFI.setDHCPoptions(DHCP_OFF);
        if (SERVER_MODE)
            WIFI.setIp(SERVER_IP);
        else
            WIFI.setIp(CLIENT_IP);
        WIFI.setNetmask(NETMASK);
        WIFI.setGW(GATEWAY);
        WIFI.setJoinMode(MANUAL);
        WIFI.setAuthKey(WPA2, AUTHKEY);
        WIFI.storeData();

        // Conexión a la red WiFi
        if (WIFI.join(ESSID)) {
            USB.println(F("\nJoined AP"));
            USB.println(F("-----"));
            USB.println(F("get IP"));
        }
    }
}

```

```

USB.println(F("-----\n"));
WIFI.getIP();
notifyByLEDS(false,3);

// Modo Servidor (sink node)
if (SERVER_MODE) {

    if (numAlarms>MAX_NUM_CLIENTS) {
        USB.println(F("ERROR -> The number of client nodes is
greater than allowed"));
        notifyByLEDS(true,0);
        USB.println(F("\nPROGRAM STOPPED -> Restart waspmote and try
again!!"));
        USB.println(F("*****"));
        delay(300000);
    }
    else {

        // Se van descartando los nodos cliente según van recibiendo
su configuración
        for (int clients = 0; clients<numAlarms; clients++) {
            while (!messageReceived) {
                sendConfig();
            }
            WIFI.leave(); // Cuando un nodo cliente recibe su
configuración, el servidor
                        // se desconecta de la red y luego repite
todo el proceso
            while (messageReceived) {
                if (WIFI.join(ESSID)) {
                    notifyByLEDS(false,3);
                    messageReceived = false;
                }
            }
        }
    }

// Modo Cliente
else {
    USB.println(F("\nWaiting for the configuration turn...\n"));
    delay(60000); // Espera al servidor: 60 segundos
    while (!messageReceived) {
        receiveConfig();
        delay(8000); // Mejora la probabilidad de conexión con el
sink node
    }
    messageReceived = false;
}

// Desconexión de la red WiFi
WIFI.leave();
}
else {
    USB.println(F("NOT Connected to AP"));
    notifyByLEDS(true,0);
}

USB.println(F("\nSETUP COMPLETED\n"));
}

```



```

    else {
        USB.println(F("ERROR -> WiFi did not initialize correctly"));
        notifyByLEDS(true,0);
        USB.println(F("\nPROGRAM STOPPED -> Restart waspmote and try again!!"));
        USB.println(F("*****"));
        delay(300000);
    }
}

// RUTINA PRINCIPAL DE EJECUCIÓN
void loop() {

    // Modo Servidor
    if (SERVER_MODE && numAlarms>0) {

        // Configuración de la alarma (asignadas de forma consecutiva)
        char settingAlarm1[11];
        strcpy(settingAlarm1,"00:00:");
        strcat(settingAlarm1,alarms[configuredAlarm]);
        strcat(settingAlarm1,":00");
        RTC.setAlarm1(settingAlarm1,RTC_ABSOLUTE,RTC_ALM1_MODE4);
        USB.printf("\nAlarm1: %s\n", RTC.getAlarm1());
        configuredAlarm++;
        if(configuredAlarm>=numAlarms)
            configuredAlarm = 0;

    }

    // Modo Cliente
    else if (!SERVER_MODE) {

        // Configuración de la alarma
        char settingAlarm1[11];
        strcpy(settingAlarm1,"00:00:");
        strcat(settingAlarm1,myAlarm);
        strcat(settingAlarm1,":00");
        RTC.setAlarm1(settingAlarm1,RTC_ABSOLUTE,RTC_ALM1_MODE4);
        USB.printf("\nAlarm1: %s\n", RTC.getAlarm1());

    }
    else {
        USB.println(F("No function expected"));
    }

    // Waspote en Modo Sleep
    USB.println(F("Waspote goes to sleep...\n"));
    notifyByLEDS(false,2);
    PWR.sleep(ALL_OFF);

    // ----->SLEEP MODE<-----

    // Configuración de módulos tras Modo Sleep
    RTC.ON();
    USB.ON();
    USB.println(F("Waspote wakes up!!"));

    // Configuración post-interrupción
    if( intFlag & RTC_INT ) {

```

```

intFlag &= ~(RTC_INT); // Limpiar flag
USB.println(F("-----"));
USB.println(F("RTC INT Captured"));
USB.println(F("-----"));

// Módulo WiFi ON
if( WIFI.ON(socket) == 1 ) {

    USB.println(F("\nWiFi switched ON\n"));

    // Modo Servidor
    if (SERVER_MODE && numAlarms>0) {

        // Recepción de datos del nodo cliente + envío de datos al
servidor HTTP
        server();
        if (messageReceived) {
            sendMessageToServer();
            messageReceived = false;
        }
    }

    // Modo Cliente
    else if (!SERVER_MODE) {

        // Envío de datos recopilados al nodo servidor
        delay(15000); // Espera para la configuración del servidor
TCP
        client();
        messageReceived = false;
    }

}
else {
    USB.println(F("Wifi did not initialize correctly\n"));
    notifyByLEDS(true,0);
}

}
}

// DESCARGA Y GUARDADO DE LA CONFIGURACIÓN DE LA RED
void downloadConfig() {

    // Conexión a la red WiFi
    if (WIFI.join(ESSID)) {

        USB.println(F("Joined\n"));
        notifyByLEDS(false,3);

        // Configuración de la consulta al host
        snprintf(body, sizeof(body), "nombre=%s", name);
        USB.println(F("Connecting to server..."));
        USB.print(F("GET: "));
        USB.println(body);

        // Consulta al host
        do {
            status = WIFI.getURL(DNS, HOST, urlConfig, body);
        }
    }
}

```

```

while (status!=1);

if (status==1) {

    USB.println(F("HTTP query OK.));
    notifyByLEDS(false,6);

    // Respuesta del host
    strcpy(config, WIFI.answer);
    USB.println(F("\nHOST ANSWER:"));
    USB.println(WIFI.answer);
    USB.println();
    numAlarms = 0;
    int actualAlarm;

    // Procesamiento y guardado de los datos recibidos
    // Ejemplo de mensaje de configuración:
    // "&nombre=COM4&minConex=2&nombre=COM3&minConex=32&"
    for (int k=0; k<(WIFI.length); k++) {
        while (config[k]=='&' && config[k+1]=='n' && config[k+2]=='o')
        {

            // Nombre del nodo cliente
            k = k + 8;
            int numCaract = 0;
            while (config[k]!='&') {
                clientsName[numAlarms][numCaract] = config[k];
                numCaract++;
                k++;
            }

            // Alarma del nodo cliente
            k = k + 10;
            if (config[k+1]=='&') {
                alarms[numAlarms][0] = '0';
                alarms[numAlarms][1] = config[k];
                k = k + 1;
            }
            else {
                alarms[numAlarms][0] = config[k];
                alarms[numAlarms][1] = config[k+1];
                k = k + 2;
            }
            actualAlarm = numAlarms;
            numAlarms++;
            USB.printf("Alarm %d: at %s minutes every hour\n",
numAlarms, alarms[actualAlarm]);
        }
    }

    USB.printf("Number of alarms to set: %d\n", numAlarms);
}

else {
    USB.println(F("\nHTTP query ERROR"));
    notifyByLEDS(true,0);
}

// Desconexión de la red WiFi
WIFI.leave();
}

```

```

else {
    USB.println(F("NOT joined"));
    notifyByLEDS(true,0);
}
}

// ENVÍO DE LOS DATOS DE CONFIGURACIÓN A LOS WASPMOTE-CLIENTES
void sendConfig() {

    // Establecimiento del servidor TCP
    if (WIFI.setTCPserver(SERVER_PORT)) {

        notifyByLEDS(false,4);
        USB.println(F("\nTCP server set"));
        USB.println(F("Listening for incoming data requests..."));

        // Tiempo de espera para la recepción de peticiones
        previous=millis();
        while( millis()-previous<SERVER_SHORT_TIMEOUT ) {

            // Lectura de mensajes de la conexión TCP
            WIFI.read(NOBL0);
            USB.print(F("w")); // Esperando peticiones

            // Procesamiento de la petición recibida
            if (WIFI.length>0) {
                for (int k = 0; k<(WIFI.length); k++) {
                    if (WIFI.answer[k]=='&' && WIFI.answer[k+1]=='n' &&
WIFI.answer[k+2]=='o') {
                        notifyByLEDS(false,5);
                        messageReceived = true;
                        strcpy(receiver, WIFI.answer);

                        // Nombre del cliente TCP
                        USB.print(F("\n\nIncoming request from: "));
                        k = k + 8;
                        int numCaracter = 0; //Nombres de máximo 4 caracteres
                        while (numCaracter<4) {
                            USB.print(receiver[k]);
                            clientName[numCaracter] = receiver[k];
                            numCaracter++;
                            k++;
                        }

                        // Búsqueda del cliente en el registro de nombres +
                        consulta de su alarma
                        int i;
                        for (i=0; i<numAlarms; i++) {
                            if ((clientsName[i][0]==clientName[0]) &&
(clientName[i][1]==clientName[1]) &&
(clientName[i][2]==clientName[2]) && (clientsName[i
][3]==clientName[3])) {
                                myAlarm[0] = alarms[i][0];
                                myAlarm[1] = alarms[i][1];
                            }
                        }

                        // Envío de la alarma correspondiente
                        USB.print(F("\nSending configuration data: "));

```

```

        snprintf(body, sizeof(body), "&minConex=%s/00", myAlarm);
        USB.println(body);
        WIFI.send(body);
        break;
    }
}

// Cancelación de la espera con la recepción de un mensaje
if (messageReceived) {
    break;
}

// Condición para evitar un 'overflow'
if (millis() < previous) {
    previous = millis();
}
}

// Fin de la conexión TCP
USB.println(F("\nClose the TCP connection (WiFi switched
OFF)\n"));
WIFI.close();
}
else {
    USB.println(F("TCP server NOT set\n"));
    notifyByLEDS(true, 0);
}
}

void receiveConfig() {

    // Conexión con el servidor TCP
    if (WIFI.setTCPClient(SERVER_IP, SERVER_PORT, CLIENT_PORT)) {
        notifyByLEDS(false, 4);
        USB.println(F("\nTCP client set"));

        // Petición de datos de configuración al nodo servidor
        USB.print(F("\nRequesting configuration data from the server: "));
        snprintf(body, sizeof(body), "&nombre=%s/00", name);
        USB.println(body);
        WIFI.send(body);

        // Espera para la recepción de los datos
        USB.println(F("Listen to TCP socket:"));
        previous=millis();
        while(millis()-previous<CLIENT_SHORT_TIMEOUT) {
            if(WIFI.read(NOBL0)>0) {

                // Chequeo de la respuesta recibida
                for (int k = 0; k<(WIFI.length); k++) {
                    if (WIFI.answer[k]=='&' && WIFI.answer[k+1]=='m' &&
WIFI.answer[k+2]=='i') {
                        USB.println(F("Data configuration received!!\n"));
                        messageReceived = true;
                        notifyByLEDS(false, 6);

                        // Alarma para la conexión periódica con el servidor
                        k = k + 10;
                        if (WIFI.answer[k+1]=='&') {

```

```

        myAlarm[0] = '0';
        myAlarm[1] = WIFI.answer[k];
    }
    else {
        myAlarm[0] = WIFI.answer[k];
        myAlarm[1] = WIFI.answer[k+1];
    }
    USB.printf("Alarm time: at %s minutes every hour\n",
myAlarm);
    break;
}
}

// Cancelación de la espera con la recepción de un mensaje
if (messageReceived) {
    break;
}

// Condición para evitar un 'overflow'
if (millis() < previous) {
    previous = millis();
}
}

if (!messageReceived) {
    USB.println(F("Data configuration not received, trying
again...\n"));
}

// Fin de la conexión TCP
USB.println(F("Close TCP socket\n"));
WIFI.close();
}
else {
    USB.println(F("TCP client NOT set"));
    notifyByLEDS(true,0);
}
}

// CONEXIÓN TCP PARA EL MODO SERVIDOR
void server() {

    // Conexión a la red Wi-Fi
    if (WIFI.join(ESSID)) {

        USB.println(F("Joined AP"));
        notifyByLEDS(false,3);
        USB.println(F("-----"));
        USB.println(F("get IP"));
        USB.println(F("-----\n"));
        WIFI.getIP();

        // Establecimiento del servidor TCP
        if (WIFI.setTCPserver(SERVER_PORT)) {

            notifyByLEDS(false,4);
            USB.println(F("\nTCP server set"));
            USB.print(F("Listening for incoming data during "));
            USB.print(SERVER_TIMEOUT);

```

```

USB.println(F(" milliseconds"));

// Tiempo de espera para la recepción de datos
previous=millis();
while( millis()-previous<SERVER_TIMEOUT ) {

    // Lectura de mensajes de la conexión TCP
    WIFI.read(NOBL0);

    // Procesamiento de los mensajes recibidos
    if(WIFI.length>0 && WIFI.answer[2]=='M') {

        notifyByLEDS(false,6);
        messageReceived = true;

        // Envío de confirmación de la recepción de los datos al
cliente TCP
        strcpy(receiver, WIFI.answer);
        WIFI.send("&OK");

        // Nombre del cliente TCP
        USB.print(F("\nIncoming message from "));
        for (int k=0; (receiver[k]!='&'); k++) {
            USB.print(receiver[k]);
            clientName[k] = receiver[k];
        }

        // Número de mensajes recibidos
        numMessages = receiver[5] - '0';
        int prov5 = 0;
        USB.printf("\nNumber of messages received: %d\n",
numMessages);

        // Guardado de los mensajes recibidos
        USB.print(F("\nMessage received: "));
        for (int k=8; k<(sizeof(receiver)); k++) {
            if (receiver[k]=='/' && receiver[k+1]=='0' &&
receiver[k+2]=='0') {
                if (numMessages>1) {
                    USB.print(F("\n\nOld message received: "));
                    k = k + 5;
                    prov5 = 0;
                    for (int j=k; j<(sizeof(receiver)-k); j++) {
                        if (receiver[j]=='/' && receiver[j+1]=='0' &&
receiver[j+2]=='0') {
                            break;
                        }
                    }
                    else {
                        USB.print(receiver[j]);
                        oldMessage[prov5] = receiver[j];
                        prov5++;
                    }
                }
            }
            break;
        }
        else {
            USB.print(receiver[k]);
            message[prov5] = receiver[k];
            prov5++;
        }
    }
}

```

```

        }
        break;
    }

    // Condición para evitar un 'overflow'
    if (millis() < previous) {
        previous = millis();
    }

}

// Fin de la conexión TCP
USB.println(F("\n\nClose the TCP connection (WiFi switched
OFF)"));
WIFI.close();

}
else {
    USB.println(F("TCP server NOT set"));
    notifyByLEDS(true,0);
}

// Desconexión de la red Wifi
WIFI.leave();
}

else {
    USB.println(F("NOT Connected to AP\n"));
    notifyByLEDS(true,0);
}

USB.println(F("*****"));
}

// CONEXIÓN TCP PARA EL MODO CLIENTE
void client() {

    // Preparación del nuevo mensaje (fecha y nivel de batería)
    snprintf(message, sizeof(message), "Datos recopilados a %s con un
%d%s de batería",
        RTC.getTime(), PWR.getBatteryLevel(), "%");

    // Conexión a la red WiFi
    if (WIFI.join(ESSID)) {

        USB.println(F("Joined AP"));
        notifyByLEDS(false,3);
        WIFI.getIP();

        // Conexión con el servidor TCP
        if (WIFI.setTCPclient(SERVER_IP, SERVER_PORT, CLIENT_PORT)) {

            notifyByLEDS(false,4);
            USB.println(F("TCP client set"));

            // Envío de mensajes al servidor
            USB.print(F("Sending data: "));
            if (numMessages>1) {
                snprintf(body, sizeof(body), "%s&%d->%s/00->%s/00",
                    name, numMessages, message, oldMessage);
            }
        }
    }
}

```



```

    }
    else {
        snprintf(body, sizeof(body), "%s%d->%s/00",
            name, numMessages, message);
    }
    USB.println(body);
    WIFI.send(body);

    // Espera para la confirmación del servidor
    USB.println(F("Listen to TCP socket:"));
    previous=millis();
    while(millis()-previous<CLIENT_TIMEOUT) {
        if(WIFI.read(NOBL0)>0) {

            // Chequeo de la respuesta recibida
            for (int k = 0; k<(WIFI.length); k++) {
                if (WIFI.answer[k]=='&' && WIFI.answer[k+1]=='O' &&
WIFI.answer[k+2]=='K') {
                    USB.println(F("OK -> Data sent correctly!!\n"));
                    messageReceived = true;
                    notifyByLEDS(false,6);
                    break;
                }
            }

        }

        // Condición para evitar un 'overflow'
        if (millis() < previous) {
            previous = millis();
        }

    }

    // Fin de la conexión TCP
    USB.println(F("Close TCP socket"));
    WIFI.close();

}

else {
    USB.println(F("TCP client NOT set"));
    notifyByLEDS(true,0);
}

// Desconexión de la red Wifi
WIFI.leave();
}

else {
    USB.println(F("NOT Connected to AP"));
    notifyByLEDS(true,0);
}

// Si el mensaje no ha sido recibido por el servidor se almacena
if (!messageReceived) {
    numMessages = 2;
    USB.println(F("\nSaving last message -> Any answer received from
TCP server"));
    for (int k=0; k<(sizeof(message)); k++) {
        oldMessage[k] = message[k];
    }
}

```

```

    }
}
else {
    numMessages = 1;
}
}

// ENVÍO DE DATOS A LA BASE DE DATOS DEL HOST
void sendMessageToServer() {

    // Configuración del módulo Wifi (conexión HTTP)
    WIFI.resetValues();
    WIFI.setConnectionOptions(HTTP);
    WIFI.setDHCPoptions(DHCP_OFF);
    WIFI.setIp(SERVER_IP);
    WIFI.setNetmask(NETMASK);
    WIFI.setGW(GATEWAY);
    WIFI.setJoinMode(MANUAL);
    WIFI.setAuthKey(WPA2, AUTHKEY);
    WIFI.storeData();

    // Conexión a la red Wifi
    if (WIFI.join(ESSID)) {

        USB.println(F("\nJoined\n"));
        notifyByLEDS(false, 3);

        // Configuración y envío al host de mensajes antiguos
        if (numMessages>1) {

            for (int k = 0; k<(sizeof(oldMessage)); k++) {
                if (oldMessage[k]==' ') {
                    oldMessage[k] = '_';
                }
            }

            snprintf(body, sizeof(body),
"nombre=%s&datos=MENSAJE_RECUPERADO:_%s", clientName, oldMessage);
            USB.println(F("Connecting to server to update old messages..."));
            USB.print(F("GET: "));
            USB.println(body);

            previous=millis();
            do {
                status = WIFI.getUrl(DNS, HOST, urlData, body);
            }
            while (status!=1 && (millis()-previous<CLIENT_TIMEOUT));

            // Respuesta del host
            if (status==1) {
                USB.println(F("HTTP query OK\n"));
                notifyByLEDS(false, 6);
                USB.println(F("HOST ANSWER:"));
                USB.println(WIFI.answer);
                numMessages = 1;
            }
            else {
                USB.println(F("\nHTTP query ERROR"));
                notifyByLEDS(true, 0);
            }
        }
    }
}

```

```

    }

    // Configuración y envío al host del último mensaje recibido
    for (int k = 0; k < (sizeof(message)); k++) {
        if (message[k] == ' ') {
            message[k] = '_';
        }
    }

    snprintf(body, sizeof(body), "nombre=%s&datos=%s", clientName,
message);
    USB.println(F("Conecting to server..."));
    USB.print(F("GET: "));
    USB.println(body);

    previous = millis();
    do {
        status = WIFI.getURL(DNS, HOST, urlData, body);
    }
    while (status != 1 && (millis() - previous < CLIENT_TIMEOUT));

    // Respuesta del host
    if (status == 1) {
        USB.println(F("\nHTTP query OK\n"));
        notifyByLEDS(false, 6);
        USB.println(F("HOST ANSWER:"));
        USB.println(WIFI.answer);
    }
    else {
        USB.println(F("\nHTTP query ERROR"));
        notifyByLEDS(true, 0);
    }
}
else {
    USB.println(F("NOT Connected to AP\n"));
    notifyByLEDS(true, 0);
}

// Configuración del módulo WiFi (conexión TCP)
WIFI.resetValues();
WIFI.setConnectionOptions(CLIENT_SERVER);
WIFI.setDHCPoptions(DHCP_OFF);
WIFI.setIp(SERVER_IP);
WIFI.setNetmask(NETMASK);
WIFI.setGW(GATEWAY);
WIFI.setJoinMode(MANUAL);
WIFI.setAuthKey(WPA2, AUTHKEY);
WIFI.storeData();
}

// NOTIFICACIONES LED
void notifyByLEDS(boolean error, int times) {
    if (error) {
        Utils.blinkRedLED(250, 5);
    }
    else {
        Utils.blinkGreenLED(500, times);
    }
}
}

```

Código PHP 5.6 del archivo "config.php"

```
1. <?php
2.
3.     $host = "mysql.hostinger.es";
4.     $usuario = "u568823178_us";
5.     $contrasena = "waspmote";
6.     $db = "u568823178_prueb";
7.
8.     if (isset($_GET["nombre"]) && !empty($_GET["nombre"])) {
9.
10.         $nombre = $_GET["nombre"];
11.
12.         $conexion = mysqli_connect($host, $usuario,
    $contrasena)
13.         or die ("Problema con el servidor");
14.
15.         mysqli_select_db($conexion, $db)
16.         or die ("Problema al seleccionar la BD");
17.
18.         $consulta = $conexion->prepare("SELECT
    admin FROM config WHERE nombre=?");
19.         $consulta->bind_param('s', $nombre);
20.         $consulta->execute();
21.         $consulta->store_result();
22.         $consulta->bind_result($admin);
23.         $consulta->fetch();
24.         $consulta->close();
25.
26.         if ($admin=="si") {
27.             $parametros = $conexion-
    >query("SELECT nombre,minConex FROM config WHERE admin='no'
    ORDER BY minConex");
28.             while($fila = $parametros-
    >fetch_array()) {
29.                 echo "&nombre=" . $fila["no
    mbre"] . "&minConex=" . $fila["minConex"];
30.             }
31.             echo "&";
32.             $parametros->close();
33.         } else {
34.             echo "&Informacion disponible solo
    para el nodo servidor";
35.         }
36.
37.         } else {echo "&Configuracion incorrecta";}
38.
39.     ?>
```

Código PHP 5.6 del archivo "dataServer.php"

```
1. <?php
2.
3.     $host = "mysql.hostinger.es";
4.     $usuario = "u568823178_us";
5.     $contrasena = "waspmote";
6.     $db = "u568823178_prueb";
7.
8.     if (isset($_GET["nombre"]) && !empty($_GET["nombre"])) {
9.
10.         $nombre = $_GET["nombre"];
11.         $datos = $_GET["datos"];
12.
13.         $repDatos = str_replace("_", " ", $datos);
14.
15.         $conexion = mysqli_connect($host, $usuario,
    $contrasena)
16.         or die ("Problema con el servidor");
17.
18.         mysqli_select_db($conexion, $db)
19.         or die ("Problema al seleccionar la BD");
20.
21.         mysqli_query($conexion, "INSERT INTO data
    (nombre,datos,hora,fecha) VALUES
    ('$nombre','$repDatos',CURTIME(),CURDATE())")
22.         or die ("Problema al insertar datos en la
    BD");
23.
24.         echo "&UPDATED - Datos actualizados";
25.
26.     } else {echo "&ERROR - Datos incorrectos";}
27.
28.     ?>
```

ANEXO IV: Registro de mediciones

A continuación, se muestra a modo de ejemplo el registro de mediciones de una de las pruebas de funcionamiento realizadas con una duración de 3 días:

id	nombre	datos	hora	fecha
1	COM6	Datos recopilados a Tue, 04/07/17, 12:02:18 con un 86% de batería	10:52:15	24/08/2017
2	COM3	Datos recopilados a Tue, 04/07/17, 12:32:18 con un 37% de batería	11:22:14	24/08/2017
3	COM6	Datos recopilados a Tue, 04/07/17, 13:02:18 con un 87% de batería	11:52:14	24/08/2017
4	COM3	Datos recopilados a Tue, 04/07/17, 13:32:18 con un 37% de batería	12:22:16	24/08/2017
5	COM6	Datos recopilados a Tue, 04/07/17, 14:02:18 con un 86% de batería	12:52:15	24/08/2017
6	COM3	Datos recopilados a Tue, 04/07/17, 14:32:18 con un 37% de batería	13:22:16	24/08/2017
7	COM6	Datos recopilados a Tue, 04/07/17, 15:02:18 con un 86% de batería	13:52:15	24/08/2017
8	COM3	Datos recopilados a Tue, 04/07/17, 15:32:18 con un 37% de batería	14:22:20	24/08/2017
9	COM6	Datos recopilados a Tue, 04/07/17, 16:02:18 con un 86% de batería	14:52:14	24/08/2017
10	COM3	Datos recopilados a Tue, 04/07/17, 16:32:18 con un 37% de batería	15:22:16	24/08/2017
11	COM6	Datos recopilados a Tue, 04/07/17, 17:02:18 con un 86% de batería	15:52:15	24/08/2017
12	COM3	Datos recopilados a Tue, 04/07/17, 17:32:18 con un 37% de batería	16:22:17	24/08/2017
13	COM6	Datos recopilados a Tue, 04/07/17, 18:02:18 con un 86% de batería	16:52:14	24/08/2017
14	COM3	Datos recopilados a Tue, 04/07/17, 18:32:18 con un 37% de batería	17:22:18	24/08/2017
15	COM6	Datos recopilados a Tue, 04/07/17, 19:02:18 con un 86% de batería	17:52:14	24/08/2017
16	COM6	Datos recopilados a Tue, 04/07/17, 20:02:18 con un 86% de batería	18:52:15	24/08/2017
17	COM6	Datos recopilados a Tue, 04/07/17, 21:02:18 con un 86% de batería	19:52:17	24/08/2017
18	COM6	Datos recopilados a Tue, 04/07/17, 22:02:18 con un 86% de batería	20:52:14	24/08/2017
19	COM3	MENSAJE RECUPERADO: Datos recopilados a Tue, 04/07/17, 21:32:18 con un 37% de batería	21:22:20	24/08/2017
20	COM3	Datos recopilados a Tue, 04/07/17, 22:32:18 con un 37% de batería	21:22:31	24/08/2017
21	COM6	Datos recopilados a Tue, 04/07/17, 23:02:18 con un 86% de batería	21:52:15	24/08/2017
22	COM3	Datos recopilados a Tue, 04/07/17, 23:32:18 con un 37% de batería	22:22:23	24/08/2017
23	COM6	Datos recopilados a Wed, 04/07/18, 00:02:18 con un 86% de batería	22:52:14	24/08/2017
24	COM3	Datos recopilados a Wed, 04/07/18, 00:32:18 con un 37% de batería	23:22:14	24/08/2017
25	COM6	Datos recopilados a Wed, 04/07/18, 01:02:18 con un 85% de batería	23:52:14	24/08/2017
26	COM6	Datos recopilados a Wed, 04/07/18, 02:02:18 con un 85% de batería	0:52:14	25/08/2017
27	COM3	MENSAJE RECUPERADO: Datos recopilados a Wed, 04/07/18, 01:32:18 con un 37% de batería	1:22:14	25/08/2017
28	COM3	Datos recopilados a Wed, 04/07/18, 02:32:18 con un 37% de batería	1:22:25	25/08/2017
29	COM6	Datos recopilados a Wed, 04/07/18, 03:02:18 con un 85% de batería	1:52:14	25/08/2017
30	COM3	Datos recopilados a Wed, 04/07/18, 03:32:18 con un 37% de batería	2:22:20	25/08/2017
31	COM6	Datos recopilados a Wed, 04/07/18, 04:02:18 con un 85% de batería	2:52:14	25/08/2017
32	COM3	Datos recopilados a Wed, 04/07/18, 04:32:18 con un 37% de batería	3:22:23	25/08/2017
33	COM6	Datos recopilados a Wed, 04/07/18, 05:02:18 con un 85% de batería	3:52:14	25/08/2017
34	COM3	Datos recopilados a Wed, 04/07/18, 05:32:18 con un 37% de batería	4:22:15	25/08/2017
35	COM3	Datos recopilados a Wed, 04/07/18, 06:32:18 con un 37% de batería	5:22:14	25/08/2017
36	COM6	Datos recopilados a Wed, 04/07/18, 07:02:18 con un 85% de batería	5:52:14	25/08/2017
37	COM3	Datos recopilados a Wed, 04/07/18, 07:32:18 con un 37% de batería	6:22:17	25/08/2017

38	COM6	Datos recopilados a Wed, 04/07/18, 08:02:18 con un 85% de batería	6:52:15	25/08/2017
39	COM3	Datos recopilados a Wed, 04/07/18, 08:32:18 con un 37% de batería	7:22:13	25/08/2017
40	COM6	Datos recopilados a Wed, 04/07/18, 09:02:18 con un 85% de batería	7:52:14	25/08/2017
41	COM3	Datos recopilados a Wed, 04/07/18, 09:32:18 con un 37% de batería	8:22:14	25/08/2017
42	COM6	Datos recopilados a Wed, 04/07/18, 10:02:18 con un 85% de batería	8:52:15	25/08/2017
43	COM3	Datos recopilados a Wed, 04/07/18, 10:32:18 con un 37% de batería	9:22:18	25/08/2017
44	COM6	Datos recopilados a Wed, 04/07/18, 11:02:18 con un 85% de batería	9:52:18	25/08/2017
45	COM3	Datos recopilados a Wed, 04/07/18, 11:32:18 con un 37% de batería	10:22:17	25/08/2017
46	COM6	Datos recopilados a Wed, 04/07/18, 12:02:18 con un 85% de batería	10:52:15	25/08/2017
47	COM3	Datos recopilados a Wed, 04/07/18, 12:32:18 con un 37% de batería	11:22:14	25/08/2017
48	COM6	Datos recopilados a Wed, 04/07/18, 13:02:18 con un 85% de batería	11:52:14	25/08/2017
49	COM3	Datos recopilados a Wed, 04/07/18, 13:32:18 con un 37% de batería	12:22:15	25/08/2017
50	COM6	Datos recopilados a Wed, 04/07/18, 14:02:18 con un 85% de batería	12:52:14	25/08/2017
51	COM6	Datos recopilados a Wed, 04/07/18, 15:02:18 con un 85% de batería	13:52:15	25/08/2017
52	COM6	Datos recopilados a Wed, 04/07/18, 16:02:18 con un 85% de batería	14:52:15	25/08/2017
53	COM6	Datos recopilados a Wed, 04/07/18, 17:02:18 con un 85% de batería	15:52:14	25/08/2017
54	COM3	MENSAJE RECUPERADO: Datos recopilados a Wed, 04/07/18, 16:32:18 con un 37% de batería	16:22:19	25/08/2017
55	COM3	Datos recopilados a Wed, 04/07/18, 17:32:18 con un 37% de batería	16:22:31	25/08/2017
56	COM6	Datos recopilados a Wed, 04/07/18, 18:02:18 con un 85% de batería	16:52:15	25/08/2017
57	COM3	Datos recopilados a Wed, 04/07/18, 18:32:18 con un 37% de batería	17:22:17	25/08/2017
58	COM6	Datos recopilados a Wed, 04/07/18, 19:02:18 con un 85% de batería	17:52:14	25/08/2017
59	COM3	Datos recopilados a Wed, 04/07/18, 19:32:18 con un 37% de batería	18:22:16	25/08/2017
60	COM6	Datos recopilados a Wed, 04/07/18, 20:02:18 con un 85% de batería	18:52:14	25/08/2017
61	COM3	Datos recopilados a Wed, 04/07/18, 20:32:18 con un 37% de batería	19:22:17	25/08/2017
62	COM6	Datos recopilados a Wed, 04/07/18, 21:02:18 con un 85% de batería	19:52:14	25/08/2017
63	COM3	Datos recopilados a Wed, 04/07/18, 21:32:18 con un 37% de batería	20:22:16	25/08/2017
64	COM6	Datos recopilados a Wed, 04/07/18, 22:02:18 con un 85% de batería	20:52:15	25/08/2017
65	COM3	Datos recopilados a Wed, 04/07/18, 22:32:18 con un 36% de batería	21:22:17	25/08/2017
66	COM6	Datos recopilados a Wed, 04/07/18, 23:02:18 con un 85% de batería	21:52:16	25/08/2017
67	COM3	Datos recopilados a Wed, 04/07/18, 23:32:18 con un 36% de batería	22:22:17	25/08/2017
68	COM6	Datos recopilados a Thu, 04/07/19, 00:02:18 con un 85% de batería	22:52:14	25/08/2017
69	COM3	Datos recopilados a Thu, 04/07/19, 00:32:18 con un 36% de batería	23:22:13	25/08/2017
70	COM6	Datos recopilados a Thu, 04/07/19, 01:02:18 con un 84% de batería	23:52:14	25/08/2017
71	COM3	Datos recopilados a Thu, 04/07/19, 01:32:18 con un 36% de batería	0:22:16	26/08/2017
72	COM6	Datos recopilados a Thu, 04/07/19, 02:02:18 con un 84% de batería	0:52:14	26/08/2017
73	COM3	Datos recopilados a Thu, 04/07/19, 02:32:18 con un 36% de batería	1:22:15	26/08/2017
74	COM6	Datos recopilados a Thu, 04/07/19, 03:02:18 con un 84% de batería	1:52:14	26/08/2017
75	COM3	Datos recopilados a Thu, 04/07/19, 03:32:18 con un 36% de batería	2:22:13	26/08/2017
76	COM6	Datos recopilados a Thu, 04/07/19, 04:02:18 con un 84% de batería	2:52:14	26/08/2017
77	COM3	Datos recopilados a Thu, 04/07/19, 04:32:18 con un 36% de batería	3:22:21	26/08/2017
78	COM6	Datos recopilados a Thu, 04/07/19, 05:02:18 con un 84% de batería	3:52:14	26/08/2017
79	COM3	Datos recopilados a Thu, 04/07/19, 05:32:18 con un 36% de batería	4:22:14	26/08/2017
80	COM6	Datos recopilados a Thu, 04/07/19, 06:02:18 con un 84% de batería	4:52:14	26/08/2017
81	COM3	Datos recopilados a Thu, 04/07/19, 06:32:18 con un 36% de batería	5:22:13	26/08/2017

82	COM6	Datos recopilados a Thu, 04/07/19, 07:02:18 con un 84% de batería	5:52:14	26/08/2017
83	COM3	Datos recopilados a Thu, 04/07/19, 07:32:18 con un 36% de batería	6:22:15	26/08/2017
84	COM6	Datos recopilados a Thu, 04/07/19, 08:02:18 con un 84% de batería	6:52:14	26/08/2017
85	COM3	Datos recopilados a Thu, 04/07/19, 08:32:18 con un 36% de batería	7:22:14	26/08/2017
86	COM6	Datos recopilados a Thu, 04/07/19, 09:02:18 con un 84% de batería	7:52:15	26/08/2017
87	COM3	Datos recopilados a Thu, 04/07/19, 09:32:18 con un 36% de batería	8:22:15	26/08/2017
88	COM6	Datos recopilados a Thu, 04/07/19, 10:02:18 con un 84% de batería	8:52:14	26/08/2017
89	COM3	Datos recopilados a Thu, 04/07/19, 10:32:18 con un 36% de batería	9:22:14	26/08/2017
90	COM6	Datos recopilados a Thu, 04/07/19, 11:02:18 con un 84% de batería	9:52:14	26/08/2017
91	COM3	Datos recopilados a Thu, 04/07/19, 11:32:18 con un 36% de batería	10:22:13	26/08/2017
92	COM6	Datos recopilados a Thu, 04/07/19, 12:02:18 con un 84% de batería	10:52:15	26/08/2017
93	COM3	Datos recopilados a Thu, 04/07/19, 12:32:18 con un 36% de batería	11:22:16	26/08/2017
94	COM6	Datos recopilados a Thu, 04/07/19, 13:02:18 con un 84% de batería	11:52:14	26/08/2017
95	COM3	Datos recopilados a Thu, 04/07/19, 13:32:18 con un 36% de batería	12:22:26	26/08/2017
96	COM6	Datos recopilados a Thu, 04/07/19, 14:02:18 con un 84% de batería	12:52:14	26/08/2017
97	COM3	Datos recopilados a Thu, 04/07/19, 14:32:18 con un 36% de batería	13:22:14	26/08/2017
98	COM6	Datos recopilados a Thu, 04/07/19, 15:02:18 con un 84% de batería	13:52:15	26/08/2017
99	COM3	Datos recopilados a Thu, 04/07/19, 15:32:18 con un 36% de batería	14:22:16	26/08/2017
100	COM6	Datos recopilados a Thu, 04/07/19, 16:02:18 con un 84% de batería	14:52:15	26/08/2017
101	COM3	Datos recopilados a Thu, 04/07/19, 16:32:18 con un 36% de batería	15:22:17	26/08/2017
102	COM6	Datos recopilados a Thu, 04/07/19, 17:02:18 con un 84% de batería	15:52:14	26/08/2017
103	COM3	Datos recopilados a Thu, 04/07/19, 17:32:18 con un 36% de batería	16:22:17	26/08/2017
104	COM6	Datos recopilados a Thu, 04/07/19, 18:02:18 con un 84% de batería	16:52:17	26/08/2017
105	COM3	Datos recopilados a Thu, 04/07/19, 18:32:18 con un 36% de batería	17:22:16	26/08/2017
106	COM6	Datos recopilados a Thu, 04/07/19, 19:02:18 con un 84% de batería	17:52:14	26/08/2017
107	COM3	Datos recopilados a Thu, 04/07/19, 19:32:18 con un 36% de batería	18:22:14	26/08/2017
108	COM6	Datos recopilados a Thu, 04/07/19, 20:02:18 con un 84% de batería	18:52:15	26/08/2017
109	COM3	Datos recopilados a Thu, 04/07/19, 20:32:18 con un 36% de batería	19:22:14	26/08/2017
110	COM6	Datos recopilados a Thu, 04/07/19, 21:02:18 con un 84% de batería	19:52:15	26/08/2017
111	COM3	Datos recopilados a Thu, 04/07/19, 21:32:18 con un 36% de batería	20:22:14	26/08/2017
112	COM6	Datos recopilados a Thu, 04/07/19, 22:02:18 con un 84% de batería	20:52:17	26/08/2017
113	COM3	Datos recopilados a Thu, 04/07/19, 22:32:18 con un 36% de batería	21:22:17	26/08/2017
114	COM6	Datos recopilados a Thu, 04/07/19, 23:02:18 con un 84% de batería	21:52:14	26/08/2017
115	COM3	Datos recopilados a Thu, 04/07/19, 23:32:18 con un 36% de batería	22:22:14	26/08/2017
116	COM6	Datos recopilados a Fri, 04/07/20, 00:02:18 con un 84% de batería	22:52:14	26/08/2017
117	COM3	Datos recopilados a Fri, 04/07/20, 00:32:18 con un 36% de batería	23:22:19	26/08/2017
118	COM6	Datos recopilados a Fri, 04/07/20, 01:02:18 con un 84% de batería	23:52:14	26/08/2017
119	COM6	Datos recopilados a Fri, 04/07/20, 02:02:18 con un 84% de batería	0:52:15	27/08/2017
120	COM3	MENSAJE RECUPERADO: Datos recopilados a Fri, 04/07/20, 01:32:18 con un 36% de batería	1:22:14	27/08/2017
121	COM3	Datos recopilados a Fri, 04/07/20, 02:32:18 con un 36% de batería	1:22:25	27/08/2017
122	COM6	Datos recopilados a Fri, 04/07/20, 03:02:18 con un 84% de batería	1:52:14	27/08/2017
123	COM3	Datos recopilados a Fri, 04/07/20, 03:32:18 con un 36% de batería	2:22:16	27/08/2017
124	COM6	Datos recopilados a Fri, 04/07/20, 04:02:18 con un 84% de batería	2:52:14	27/08/2017
125	COM3	Datos recopilados a Fri, 04/07/20, 04:32:18 con un 35% de batería	3:22:14	27/08/2017

126	COM6	Datos recopilados a Fri, 04/07/20, 05:02:18 con un 84% de batería	3:52:14	27/08/2017
127	COM3	Datos recopilados a Fri, 04/07/20, 05:32:18 con un 35% de batería	4:22:16	27/08/2017
128	COM6	Datos recopilados a Fri, 04/07/20, 06:02:18 con un 83% de batería	4:52:14	27/08/2017
129	COM3	Datos recopilados a Fri, 04/07/20, 06:32:18 con un 35% de batería	5:22:14	27/08/2017
130	COM6	Datos recopilados a Fri, 04/07/20, 07:02:18 con un 83% de batería	5:52:14	27/08/2017
131	COM3	Datos recopilados a Fri, 04/07/20, 07:32:18 con un 35% de batería	6:22:17	27/08/2017
132	COM6	Datos recopilados a Fri, 04/07/20, 08:02:18 con un 83% de batería	6:52:14	27/08/2017
133	COM3	Datos recopilados a Fri, 04/07/20, 08:32:18 con un 35% de batería	7:22:15	27/08/2017
134	COM6	Datos recopilados a Fri, 04/07/20, 09:02:18 con un 83% de batería	7:52:14	27/08/2017
135	COM3	Datos recopilados a Fri, 04/07/20, 09:32:18 con un 35% de batería	8:22:16	27/08/2017
136	COM6	Datos recopilados a Fri, 04/07/20, 10:02:18 con un 83% de batería	8:52:14	27/08/2017
137	COM6	Datos recopilados a Fri, 04/07/20, 11:02:18 con un 83% de batería	9:52:14	27/08/2017
138	COM3	MENSAJE RECUPERADO: Datos recopilados a Fri, 04/07/20, 10:32:18 con un 35% de batería	10:22:17	27/08/2017
139	COM3	Datos recopilados a Fri, 04/07/20, 11:32:18 con un 35% de batería	10:22:28	27/08/2017